

REPORT DOCUMENTATION PAGE	
---------------------------	--

AFRL-SR-BL-TR-98-

02492

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE May 1998	3. REPORT TYPE AND DATES COVERED Final Technical Report, 1/15/95 - 1/14/98	
4. TITLE AND SUBTITLE TIME SENSITIVE CONTROL OF AIR COMBAT OPERATIONS			5. FUNDING NUMBERS F46920-95-1-0134	
6. AUTHOR(S) Alexander H. Levis			8. PERFORMING ORGANIZATION REPORT NUMBER GMU/C3I-201-R	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Center of Excellence in Command, Control, Communications, and Intelligence George Mason University Fairfax, Virginia 22030-4444				
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFOSR/NM Air Force Office of Scientific Research Bolling Air Force Base, DC 20332-6448			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES				
12a DISTRIBUTION/AVAILABILITY STATEMENT UNlimited			12b DISTRIBUTION CODE DISTRIBUTION STATEMENT A Approved for public release; Distribution Unlimited	
13. ABSTRACT (Maximum 200 words) Research on the replanning problem in Air Combat operations is reported. The approach is based on an extension of temporal logic in which both intervals and time points are considered. The Point-Interval Temporal Logic (PITL) axiomatic formulation leads to algorithms that unify the temporal statements to form a Point Graph with an underlying Petri Net. Petri Net based algorithms are then used to analyze the graph to determine its properties. An inference engine is then used to determine whether replanning solutions exist. Two versions of the approach have been implemented. TEMPER 1 is used for a qualitative analysis of the temporal statements (sequencing of events) while TEMPER 2 accommodates time stamps and interval durations for a quantitative analysis. Both algorithms deal with the Single Time Line, Single Future case. Two examples are presented to illustrate the approach.				
14. SUBJECT TERMS Discrete Event Systems, Temporal Logic, Colored Petri Nets, Collaborative Planning			15. NUMBER OF PAGES 102	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION UNCLASSIFIED	18. SECURITY CLASSIFICATION UNCLASSIFIED	19. SECURITY CLASSIFICATION UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

19980615 123

May 1998

GMU/C3I-201-R

**CENTER OF EXCELLENCE IN C3I
GEORGE MASON UNIVERSITY
Fairfax, VA 22030**

**TIME SENSITIVE CONTROL OF
AIR COMBAT OPERATIONS**

**FINAL TECHNICAL REPORT
Contract No. F49620-95-1-0134
for the period
15 January 1995 to 14 January 1998**

Submitted to:

**DR. NEAL D. GLASSMAN
AFOSR/NM
Mathematical and Computer Sciences Directorate
Air Force Office of Scientific Research
Bolling AFB, DC 20332-6448**

Prepared by:

**Alexander H. Levis,
*Principal Investigator***

May 3, 1998

DTIC QUALITY INSPECTED 3

TABLE OF CONTENTS

1.	INTRODUCTION	5
1.1	Project Objectives	5
1.2	Project Description	5
2.	MODELING OF TIME SENSITIVE AIR COMBAT OPERATIONS USING COLORED PETRI NETS	7
2.1	Introduction	7
2.2	Use of Eagle Vision in Air Combat Operations	8
2.3	Configuration Description	10
2.4	Colored Petri Net Model Description	11
2.5	Conclusion	22
3.	ADAPTATION OF RESULTS FROM TEMPORAL LOGIC	23
3.1	Introduction	23
3.2	Mathematical Model	24
4.	TEMPORAL LOGIC / PETRI NET METHODOLOGY: TEMPER1	35
4.1	Single Time Line Single Future (STSF)	35
4.2	System Verification	40
4.3	The TL/PN Inference Engine (TIE)	47
4.4	The Algorithm	51
4.5	A Simple Example	54
4.6	TEMPER1: Software Implementation of the Algorithm	56
5.	TEMPORAL LOGIC / PETRI NET METHODOLOGY: TEMPER2	64
5.1	Introduction	64
5.2	TEMPER2	66
6.	APPLICATION: COLLABORATIVE AIR TASK PLANNING	82
6.1	Air Task Planning	82
6.2	Air Task Planning Scenario	83
6.3	Summary	99
7.	CONCLUSION	100
	REFERENCES	101

LIST OF FIGURES

Figure 1	Operations of Eagle Vision	9
Figure 2	Centralized Configuration	11
Figure 3	Colored Petri Net Model Structure	12
Figure 4	Global Declaration Node	13
Figure 5	Higher Level Page	15
Figure 6	Model of the Wing	17
Figure 7	Model of the Communications Network	18
Figure 8	Model of the Squadron	19
Figure 9	Model of the Ground Control Station	21
Figure 10	Model of the Satellite	22
Figure 11	Temporal Relations	27
Figure 12	Measure of Relative Length of Two Intervals	28
Figure 13	Algebraic Description of Temporal Relations between Intervals	30
Figure 14	String Representation of Temporal Relations	37
Figure 15	Point Graph Representation of Temporal Relations	46
Figure 16	Redundancy in PG representation	52
Figure 17	An Overall View of the Methodology	59
Figure 18.	TEMPER1 Graphical User Interface	59
Figure 19	Point Graph Generated by TEMPER1	60
Figure 20	Unified Point Graph Generated by TEMPER1	60
Figure 21	Re-Arranged Unified Point Graph	60
Figure 22	Connectivity Matrix	61
Figure 23	"Q&A" Box	61
Figure 24	"Q&A" Error Message	62
Figure 25	Correct Query Entry	62
Figure 26	TEMPER1 Response to Query	62
Figure 27	TEMPER1 Window When Save&Exit Is Executed	63
Figure 28	TEMPER1 Reminder Window Prior To Quitting	63
Figure 29	TEMPER1 Window When Re-Starting A Problem	64
Figure 30	Topology of Temporal Systems	65
Figure 31	Timed Point Graph Representation of a Temporal Situation	69

Figure 32	Unified PG Representation for Example 17	71
Figure 34	Branch Folding	74
Figure 35	Join Folding	74
Figure 36	Inconsistent Case Found During Folding Process	76
Figure 37	Unified PG for Example 17	76
Figure 38	Partially Folded TPG for Example 17	77
Figure 39	Timed Point Graph for Example 18	76
Figure 40	Black Hole Effect	78
Figure 41	Inconsistent Path Lengths	79
Figure 42	Point Graph Generated by TEMPER	85
Figure 43	Unified Point Graph of SOF Capability	85
Figure 44	Incidence Matrix Created by TEMPER	86
Figure 45	Ordinary Petri Net Representation of Incidence Matrix	86
Figure 46	Point Graph of SOF Availability and Mission Requirements	88
Figure 47	Unified Point Graph of SOF Windows of Capability and the Mission Window of Opportunity	89
Figure 48	Incidence Matrix Output of TEMPER	89
Figure 49	Ordinary Petri Net Representation of Incidence Matrix	90
Figure 50	Unified Point Graph EC Assets With individual Time Scales	92
Figure 51	Point Graph of Combined SOF and EC Windows of Capability	94
Figure 52	Unified Point Graph	95
Figure 53	TEMPER generated Incidence Matrix	95
Figure 54	Point Graph of the Final Solution to the Planning Problem	97
Figure 55	Unified Point Graph of the Final Solution	97
Figure 56	Incidence Matrix of the Final Solution	97
Figure 57	Alternative Point Graph of Final Planning Solution	98
Figure 58	Petri Net of the Incidence Matrix of the Final Planning Solution	99
Figure 59	Petri Net Converted to Final Plan	99

1. INTRODUCTION

1.1 Project Objective

The objective of this project, as described in the proposal, is threefold:

- Model time sensitive air combat operations using discrete event system models (such as Colored Petri Nets) which have been extended to include explicit representation of the times and intervals associated with events;
- Adapt results from temporal logic theories as the mathematical basis for determining windows of capability and windows of opportunity and graft this formalism to the discrete event formulation;
- Develop algorithms for manipulating windows of capability as a means of influencing the appearance and duration of windows of opportunity, and algorithms for exploiting such windows

1.2 Project Description

Two key characteristics of air combat operations are that many discrete events occur that change the "state" of the system, and that the interaction of these events establishes time intervals during which certain actions are possible. One can conceptualize the dynamic problem of the planner as the creation of advantageous *windows of capability*, and that of the controller or dynamic battle manager as the exploitation of those windows, i.e., the creation of *windows of opportunity*. The window of capability is formed by the dynamics of the discrete event system (which are affected by the plans generated by the planning cell - the ATO) while the window of opportunity depends, in addition, on the admissible actions - the systems available to the controllers during a window of capability.

The model used for the mathematical representation of the dynamics of discrete event systems is Hierarchical Colored Petri Nets (CPN). This CPN model incorporates a formal model of time that supports the correct modeling of concurrent and asynchronous operations. It is also a flexible model in the sense that one can augment it to incorporate an axiomatic model of temporal logic. A particular temporal logic model was selected, Allen's logic, and enhanced it to include constructs necessary for the control of discrete event systems and for incorporating it in the CPN formulation.

Then, using this combined model, a series of algorithms for the determination of windows of capability and windows of opportunity were derived. The temporal relations have been expressed explicitly in the CPN model (with the use of logical subnets that represent the time relationships) so that net algorithms based on the structure of the net can be used to analyze the behavioral properties of the model. Once the model has been verified, these relationships can be folded resulting in an equivalent simpler net, but with more complex annotations.

Two versions of the algorithm, TEMPER, have been developed. In the first one, only sequencing information is used to determine the appropriate windows. In the second version, time information (time stamps and durations) is included. A final step in this three year effort would have been to develop algorithms for exploiting the windows of opportunity as the means of determining the set of controls or actions. However, this work has been started but not completed during this reporting period. The general problem of multiple time lines and multiple time futures that arises when both time points and time intervals are used requires further work. When the third version of the algorithm is obtained, then the model can be exercised to test various alternatives and estimate values for the measures of performance.

The work carried out in the three year period was organized into the following tasks:

Task 1: Model time sensitive air combat operations using Colored Petri Nets

Model time sensitive air combat operations using Colored Petri Nets which have been extended to include explicit representation of the times and intervals associated with events. Focus on the planning operations at the theater level (all echelons) and at the control of air operations by AWACS and CRC controllers. Use the CPN model as the testbed for testing the algorithms for control strategies developed in Task 3 and for illustrating the relevance of the basic research effort to Air Force operational problems.

Task 2: Adapt results from temporal logic theories

Adapt results from temporal logic theories and construct the Petri net of an axiomatic system appropriate for determining windows of capability and windows of opportunity. Use theorem proving techniques and the rule base verification and validation methodology of Zaidi and Levis (1997) to develop fast, efficient algorithms.

Task 3: Develop algorithms for manipulating windows of capability

Approach the problem of time sensitive control of discrete event systems by developing algorithms for manipulating windows of capability as a means of influencing the appearance and duration of windows of opportunity, and algorithms for exploiting such windows.

Task 4: Document and disseminate the results of the research

Document the results of the research in the form of theses, technical reports, and journal papers. Present the results of the work in technical meetings. Submit progress and other reports to AFOSR in accordance with grant requirements.

Because this work has been carried out by Graduate Research Assistants pursuing the Master of Science and Ph. D. Degrees, the tasks were divided into individual projects that are appropriate for theses at the master's and Ph.D. levels. Individual students were assigned to each project under the supervision of the principal investigator who acted also as the thesis advisor. On occasion, projects are assigned to postdoctoral fellows or research faculty.

2. MODELING OF TIME SENSITIVE AIR COMBAT OPERATIONS USING COLORED PETRI NETS

The modeling of time sensitive Air Combat Operations led to two subtasks in an effort to create a test bed for evaluating the results from temporal logic. Furthermore, to illustrate the applicability of the methodology described in Task 2, an example addressing the problem of collaboration for air combat operations planning has been developed; this example is described in Section 6.

2.1. Introduction

A simple model of Air Combat Operations has been developed that can be used to test the concepts of windows of opportunity and windows of capability. This model focuses on the reception and distribution of the ATO by a Wing, detailed mission preparation by Squadrons, and the incorporation of external up-to-date images provided by satellite. This last feature is made possible by the use of an Eagle Vision system (MATRA CAP Systèmes, 1994) at the wing level. Eagle Vision consists of a transportable station and antenna, that can be deployed anywhere and designed to provide satellite imagery directly to units on the field. Reception of images at the lower

levels of planning and execution can trigger dynamic replanning of the missions, one of the concepts for which the testbed is being designed.

2.2 Use of Eagle Vision in Air Combat Operations

Eagle Vision was a project sponsored by the Foreign Comparative Testing Program in the Office of the Secretary of Defense, which procures off-the-shelf systems from non-US companies for performance evaluation in response to Department of Defense requirements. The aim of Eagle Vision is to provide a near real-time SPOT satellite imaging capability under battlefield commander control for use in mission planning and rehearsal, topographic evaluation, and intelligence analysis. The Eagle Vision system consists of a Data Acquisition Segment (DAS) provided by MATRA CAP Systemes in France, and a Data Integration Segment developed by the Environmental Research Institute of Michigan (ERIM). The Eagle Vision DAS is a transportable station capable of tracking remote sensing satellites, acquiring and processing the telemetry, and delivering images on Computer Compatible Tape (CCT). Originally designed for SPOT, the station can process Landsat data and can be upgraded for receiving ERS-1 data.

Wings and squadrons may need detailed and up-to-date information about the targets, such as Battle Damage Assessment (BDA), to plan the missions in detail. Knowing whether the enemy has recently set up new defense capabilities for a specific target may be of critical importance for the successful completion of a mission. Eagle Vision can provide detailed satellite images at the wing/squadron level. Its main feature is that it substantially reduces the delay associated with the feedback loop concerning the status of targets. During the Gulf war, images were processed and analyzed in CONUS before being forwarded to the warfighters in the theater of operations, often too late to be of any operational use. Installing Eagle Vision close to the theater of operations shifts the analysis of images to the warfighters who can focus their efforts on the targets and areas of interest to them.

Figure 1 shows schematically the operation of a deployed Eagle Vision system. The Eagle Vision has the capability to transmit by fax any request for images to a Ground Control Station (GCS) that controls the operations of the satellite. From this request, the operator at the GCS generates a programming command that specifies which picture to take and where/when to send it. This command is radio transmitted to the satellite when it is in transmitting range of the GCS. The satellite is then tasked according to this command. When the satellite is above the area of interest, it rotates its observation lens to take the picture which is then ready to be transmitted to the Eagle

Vision on the surface of the Earth. If the Eagle Vision is close enough to the area of interest, the image can be downloaded just after being taken. This configuration is called direct connection (prise directe). If the Eagle Vision is far from the area of operations, the satellite must wait to come in transmitting range before sending the data.

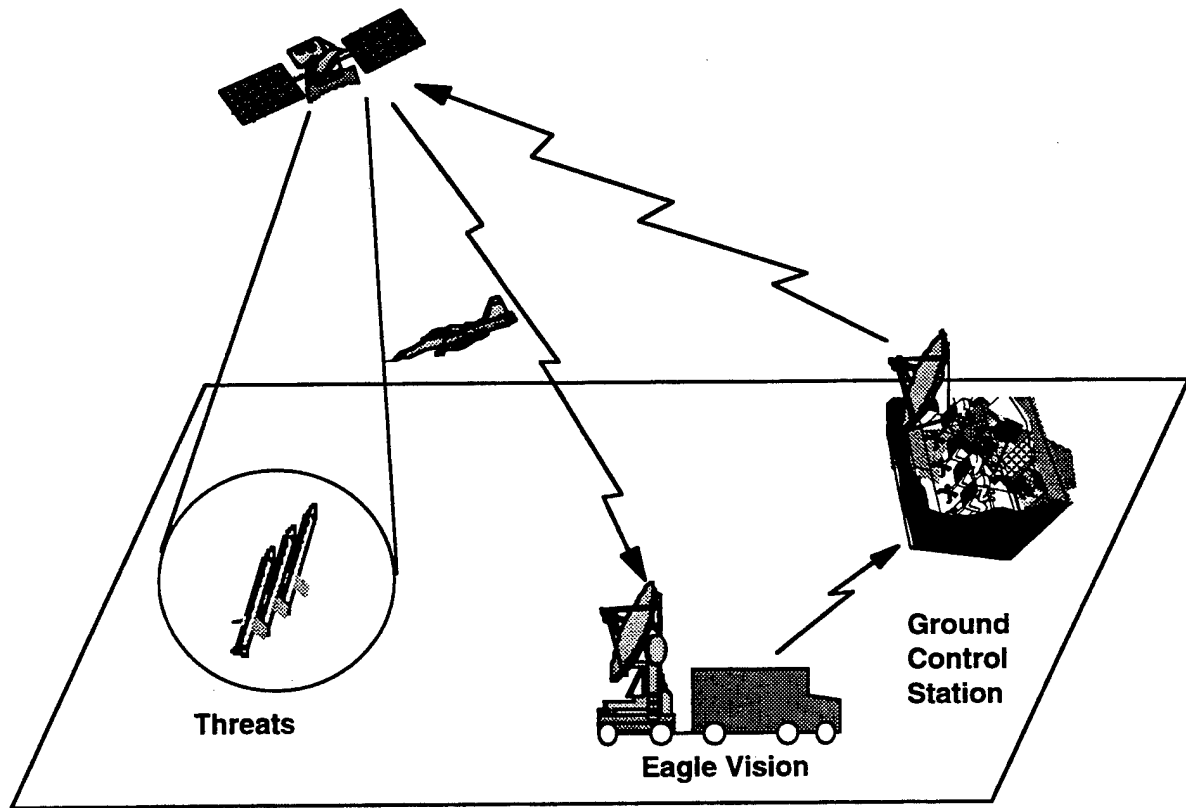


Figure 1 Operations of Eagle Vision

The satellite system supporting the Air Combat Operations system has been designed to be able to take a picture (either photo, infra-red or radar, and to download it to an Eagle Vision station within 24 hours: the squadrons receiving the ATO at the beginning of the second day (mission preparation phase) of the Air Combat Operation process must be able to generate a request for satellite imagery and to get a response before the third day begins, which is the execution phase. The satellite system consists of two satellites located on the same orbital plane with an offset of half a period. The satellites have the same orbital characteristics as the SPOT satellites (altitude 832 km, inclination angle of the orbital plane of 98.7°) but have enhanced technical capabilities to satisfy the requirements of providing images within 24 hours. Such enhanced capabilities include

multi sensing (photo, infra red, radar), sufficient resolution of images, increased maximal lateral rotation of the sensor lens, and capability to download data on Earth at any time.

2.3 Configuration Description

Two configurations of Eagle Vision have been modeled, but only one is described here. The selected configuration is a centralized one where an Eagle Vision is installed at the Wing. The alternate configuration is to install an Eagle Vision at each squadron. The reason behind the selection of the centralized configuration is that the wing seems to have sufficient authority and responsibility to initiate dynamic replanning of missions. An Eagle Vision installed at the wing level would require the existence of a local database of satellite images that could be accessed by the different squadrons. This may create several problems. The first one is an access problem. The operator at the squadron level would have to browse the database to get the images he wants. Because of the large amount of data, this process would be time consuming. A rigorous access protocol should be defined to allow all squadrons to get the data they want in a reasonable amount of time. A related problem is the support communications network that will be needed to allow the transfer of large amounts of data from the wing to the squadrons. There will not be any problem if the wings and the squadrons are close to each other and are linked by a fiber optics local area network. However, this problem could be critical if the wing and the squadrons are distant from each other and have to rely on radio or satellite communications to transfer data.

As shown on Figure 2, requests to get an image of a given area from the satellite will be sent by the squadron to the Eagle Vision operator at the wing level. This operator will have to sort the requests, identify the redundant ones, and resolve possible conflicts. Once this is done, the Eagle Vision operator will forward the request to the appropriate Ground Control Station (GCS) so that the satellite is tasked accordingly. The Eagle Vision operator will have a role similar to the anchor desk of the Copernicus Architecture (Perdu et al., 1994), that establishes the interface between non-organic sensors and the tactical operators on the battlefield. When the satellite has taken pictures and is in transmission range of the Eagle Vision, the images are downloaded to the wing, and are then forwarded to the squadrons that requested them.

2.4 Colored Petri Net Model Description

Model Structure

The Colored Petri Net model has been developed using Hierarchical Colored Petri Nets so that independent modules specified in the form of substitution transitions can be constructed and connected on the upper level page. The resulting model structure is depicted on Figure 3. Five independent model pages have been constructed to model the different parts of the Air Combat Operations using Eagle Vision. These model pages are described further in this section.

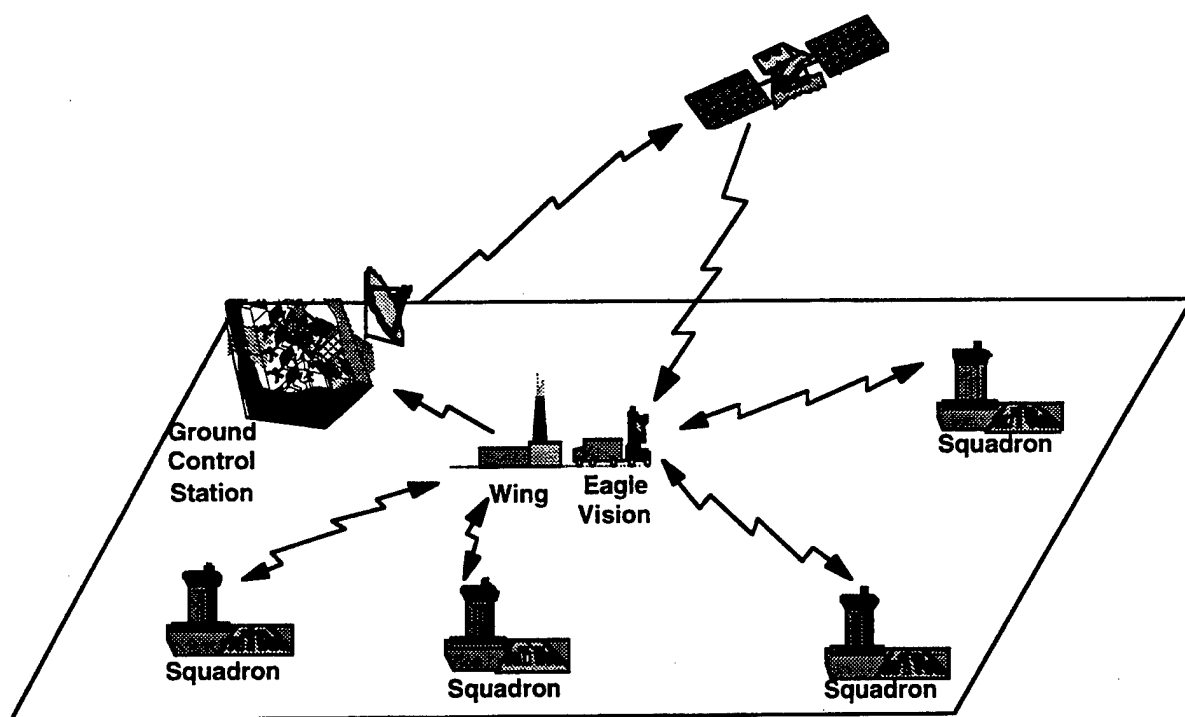


Figure 2 Centralized Configuration

Global Declaration Node

The Global Declaration Node of the model is displayed in Figure 4. It contains the declaration of the color sets and variables used in the model. The first part specifies the tokens exchanged between the object instances. The second part defines the characteristics of the satellite.

In the first part, the color set “Coord” is a real color set used for the position of the points of interest on the surface of the Earth and for the initial parameters of the position of the satellites.

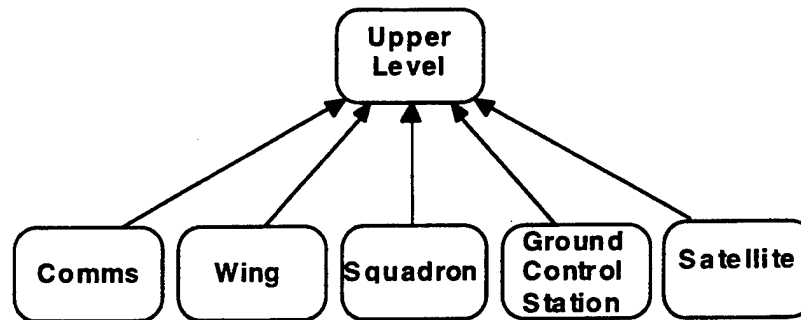


Figure 3 Colored Petri Net Model Structure

“Position” is a product color set of the color set “Coord” to indicate the latitude and longitude of the points of interest. The color set “MissionNb” is used to indicate the ID number of the mission to be carried out; “mnbn” is a variable defined for this color set. The color set “Config” has only two colors, “cent” and “decent,” for centralized and decentralized, and is used for the specialization of the class Squadron. “conf” is a variable defined for this color set. The color set “Squad” (with variable “sqnb”) is an integer color set used to identify the squadrons. The wing is identified by the color 0. “SquadData” is a composite color set which specifies the data related to each squadron. It is the product of the color set “Squad” (id number of the squadron,) and the color set “Position” which specifies where the images should be downloaded. that is the location of the wing. The color set “TimePoints” with variables “t1” and “t2” is used to specify some points in time and is mainly used for the collection of time-related data. The missions to be carried out by the squadrons are specified using the color set “Mission.” It is a product of the color sets indicating the id number of the mission (color set “MissionNb”), the position of the target (color set “Position”) the squadron that will carry out this specific mission (color set “Squad”) and when it has to take place (color set “TimePoint”). “ATO” is the list of missions and is specified as a list of color “Mission.” The color set “Request” is used to represent the requests for images by the squadrons. It is a product color set indicating which squadron generated the request (color set “Squad”), the location of the image to be taken (color set “Position”) and, where the image should be downloaded (color set “Position”). The command sent by the Ground Control station to the satellites is modelled with the color set “SatCmd” (variable “sc”) which is a product color set indicating the locations of the images to be taken and where to download the data. “DataCol” is the

timed version of the color set “SatCmd” and is used for data collection indicating when specific events ocured.

```

color Coord = real;
color Position = product Coord*Coord; (* (Latitude,Longitude) *)
color MissionNb = int; (* ID number of the mission *)
var mnb : MissionNb;
color Squad = int; (* ID number of the squadron 0 = Wing *)
var sqnb : Squad;
color SquadData = product Squad * Position;
color TimePoint = int; (* point in time *)
var t1,t2 : TimePoint;
color Mission = product MissionNb*Position*Squad*TimePoint;
color ATO = list Mission;
var mls : ATO;
color Request = product Squad*Position*Position;
color SatCmd = product Position*Position;
var sc : SatCmd;
color DataCol = SatCmd timed;
color SortReq = product Position * Position;
color Image = product Position * TimePoint;
var im1 : Image;
color MsgType = union mss:Mission + rq:Request + im:Image;
var mt : MsgType;
color CommRes = with Res timed;
color Size = int;
var sz : Size;
color Msg = product Squad * Squad * MsgType * Size timed; (* addressee, adresser,Type and Content, size *)
color UT = with ut timed;
var oslat,oslon, nslat,nslon,ilat,ilon,rlat,rlon, slat,slon,glat,glon,fi,teta : Coord;
var ipos,rpos : Position;
fun tan(x) = sin(x)/cos(x);
fun arcsin(x) = if x=0.0 then 0.0 else 2.0 * (arctan((1.0-(sqrt(1.0-(x*x))))/x));
val K=37800.0;
val Re = 6370.0;
val h = 832.0;
val pi = 2.0 * arcsin(1.0);
val a = 98.7*pi/180.0;
val T = 2.0*pi*exp(1.5*ln(Re+h))/K;
val ws = 2.0*pi/T;
val wt = 2.0*pi/1440.0;
fun rem(x:real,y:real) = let val z = x/y in x - (real(floor(z))*y) end;
fun atan2(0.0,y) = if y>0.0 then pi/2.0 else if y=0.0 then
..0 else -pi/2.0
| atan2(x,y) = let val ata=arctan(y/x) in if x >0.0 then ata else if y>0.0 then pi+ata else ata-pi end;
fun posit(t,fi:real,teta:real) =
(rem(180.0+(180.0/pi*(atan2(cos((ws*t)+(pi*fi/180.0)),(sin((ws*t)+(pi*fi/180.0))*cos(a))-((wt*t)+
(pi*teta/180.0)))), 360.0)-180.0, 180.0/pi*arcsin((sin((ws*t)+(pi*fi/180.0))*sin(a)));
fun newposit(tt,fi,teta) = posit(real(tt),fi,teta);

```

Figure 4 Global Declaration Node

The color set "Image" (variable "im1") is also a product color set to indicate the location of the center point of the image and the date at which the image was taken. The type of messages exchanged between the wing and the squadrons is specified with the color set "MsgType" (variable "mt"). It is a union color set to indicate that the message can be either a mission (color set "Mission"), a request (color set "Request"), or an image (color set "Image"). The enumerated and timed color set "CommRes" represents the resources of the communications network connecting the wing and the squadrons. The integer color set "Size" is used to specify the size of the message. Finally, the color "Msg" is used to represent the message and is the product of four color sets. The first color set "Squad" specifies the addressee of the message, the second color set "Squad" specifies the sender; the color set "MsgType" is used to specify the type and the content of the message, and "Size" the size of the message. "UT" is a timed color set used in the "Satellite" page to trigger the computation of the coordinates of the satellites in each time step of the execution. Finally, several "Coord" and "Position" variables are defined to be used in different parts of the model.

The second part specifies the different functions and parameters that deal with the orbital mechanics of the satellite. K is the constant equal to \sqrt{GM} where G is the universal constant of gravitation and M is the mass of the Earth. R_e is the radius of the Earth in km., h is the altitude of the satellite, π is the constant π (3.1415), a is the inclination of the orbital plane of the satellite in radians, T is the period of revolution of the satellite, ω_s is the angular speed of the satellite, and ω_e is the angular speed of the rotation of the earth. The function $\text{rem}(x,t)$ returns the remainder of the division of x by y ; atan2 is the quadratic arctangent. Finally, the functions "posit" and "newposit" implement the trajectory of the satellite around the Earth. Note that this model can be used to study different types of satellites having a circular orbit around the Earth: only the values of the constants h (altitude) and a (inclination) need to be modified.

Model of the Upper Level

The model of the upper level is shown on Figure 5. Every transition on this page is a substitution transition that refers to different instances of the page of the same name. These transitions correspond to objects that are different instances of the classes represented by the page described further in this section. The wing receives the ATO once a day and transmits the mission to the different squadrons through the communications represented by the substitution transition "Local Comms." The place of color set "Local Comms" specifies the number of communications

resources available. The model considers four squadrons. The data specific to the squadrons are specified by the initial markings of the places of color set "SquadData." These data are (1) the ID number of the squadrons, (2) the configuration ("cent") in which the instance is used and, (3) the location where the data should be downloaded. In the centralized configuration, the Eagle Vision is installed at the wing level and the four squadrons have the same downloading location: the wing located at 46.75° East and 24.6° North. The squadrons address their requests for images to the wing through the same communications network.

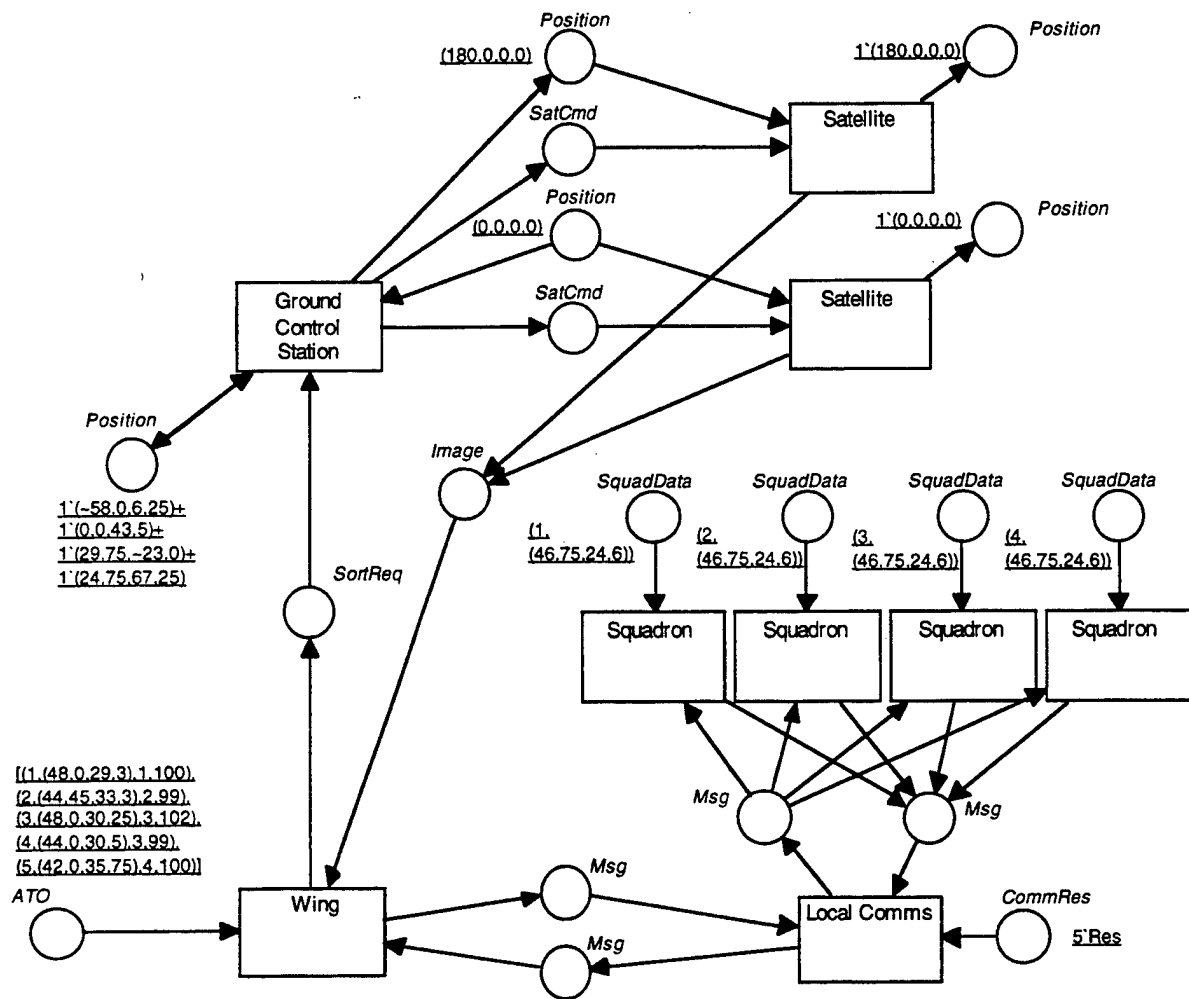


Figure 5 Higher Level Page

The wing sorts the requests and sends them to the Ground Control Stations by putting the appropriate tokens in the place of color set "SortReq." The model considers four Ground Control

Stations distributed on the surface of the Earth, as indicated by the four tokens of the the initial marking of the place of color set "Position" forming a self loop with the substitution transition "Ground Control Station." These four ground control stations are the ones used for SPOT and are located in Kourou in French Guyana (58.0 W, 6.25 N), in Aussaguel in France (0.0, 43.5 N), in Hartebeesthoek in South Africa (23.75 E, 23.0 S) and in Kiruna in Sweden (24.75 E, 67.25 N). Satellite commands are sent to the satellites when they are in transmitting range of the Ground Control Stations by putting tokens in the places of color set "SatCmd." The model considers two satellites, the characteristics of the trajectories being specified by the initial marking of the places of color set "Position": the places connected both to the transitions "Satellite" and "Ground Control Station" indicate the current position of the satellite while the places forming only a selfloop with the substitution transition "Satellite" indicate the initial parameters of the trajectory (longitude of the orbital plane at time $t = 0$ and location of the satellite on the orbital plane at time $t = 0$). Pictures are taken by the satellite according to the commands and are downloaded to the wing by putting tokens in the place of color set "Image," input to the substitution transition "Wing."

Model of the Wing

The model of the wing is displayed on Figure 6. The wing is responsible for three main activities. The first one is to disseminate the ATO received from the higher planning level. The ATO is received by the Wing when a token appears in the port place of color set "ATO." The transition "Receive ATO" fires and the missions are sent to the different squadrons when the transition "Distribute ATO" fires: for each mission defined in the ATO, a token "Msg" is generated addressed to the squadron responsible for the execution of this particular mission.

The second activity performed by the Wing is the handling of requests for images generated by the squadrons. A request is received when a "Msg" token of type request appears in the input port place of color set "Msg," enabling the transition "Process Request." When this transition fires, a "SortReq" token is generated in the output port place of color set "SortReq" and the request is also kept in memory to forward received images to the squadrons.

The third activity is the handling of images. A data base of images represented by the place of color set "Image" and named "DB" is updated when the Eagle Vision at the wing level receives images from the satellites (a token appears in the input port place of color set "Image") by firing of the transition "Receive Image." This image Database is accessed by the squadrons when an image query token appears in the input port place "Msg," enabling the transition "Receive query for

Image DB.” The response to the query takes the form of an image message represented by a token “Msg” of type Image generated in the output port place of color set “Msg.” If there is no image, the image has the form (ipos,0) and the size of the message is 10. If an image is available (the time

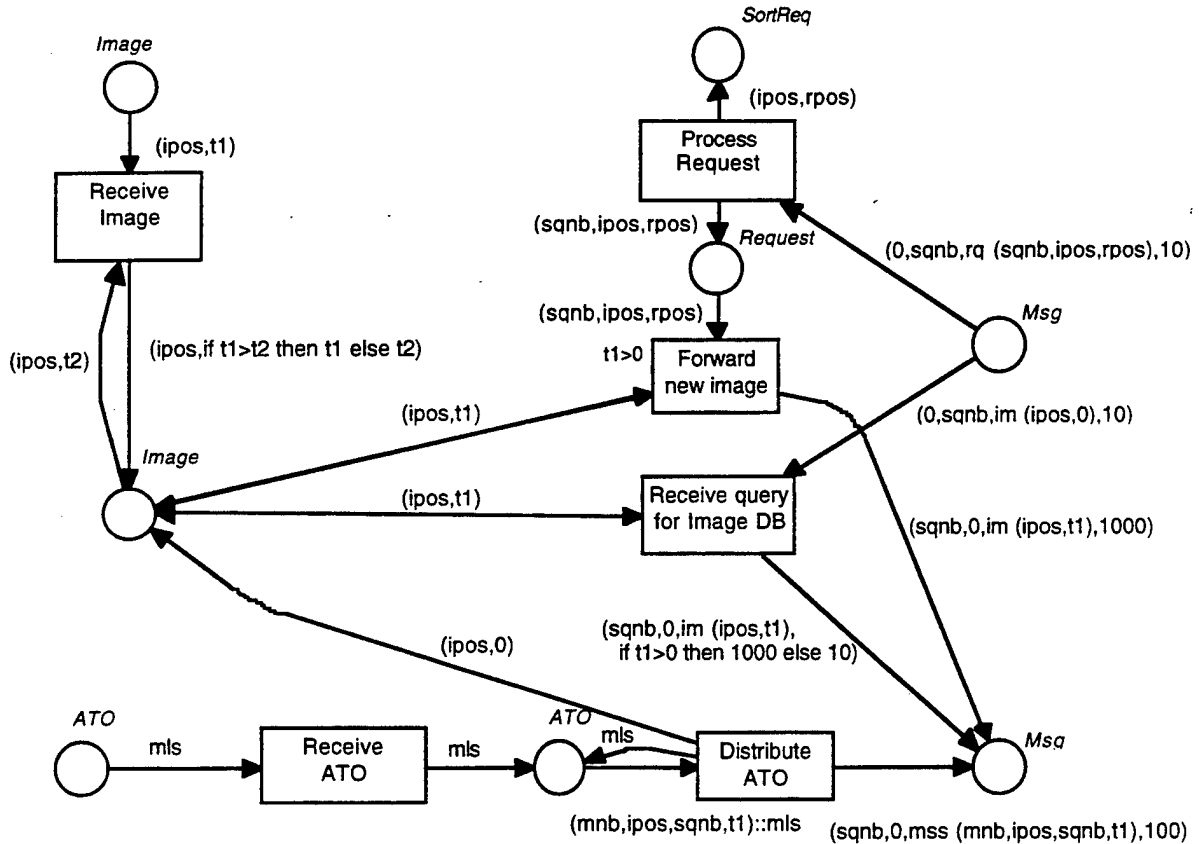


Figure 6 Model of the Wing

attribute is larger than 0), the size of the message is 1,000. Finally, the transition “Forward new image” fires when an image being requested has been put in the image database: a token “Msg” of type image is generated in the output port place of color set “Msg” addressed to the squadron that generated the request for this particular image.

Model of the Communications Network

The page model of the communications network connecting the wing to the squadrons is shown in Figure 7. This is a simple model that addresses the competition for scarce communications resources and induces a delay for the transmission messages.

The model is symmetrical: the top part models the transmission of messages from the wing to the squadrons, while the bottom part models the transmission of messages from the squadrons to the wing. A message appearing in either of the input port places of color set "Msg" requires a number of resources which is a function of its size as modeled by the arc expressions connecting the place of color set "CommRes" to the transitions: if the size of the message is less than or equal to 10, one resource is required, otherwise two resources are necessary. The delay to transmit the messages is modeled in the time region of the transition. It takes 1 unit of time (corresponding to 1 minute) to transmit a message the size of which is smaller than 11, 2 units of time for messages with size between 11 and 100, and 5 units of time for messages larger than 100. After the delay, the token "Msg" is generated in either of the output port places and the resources are given back.

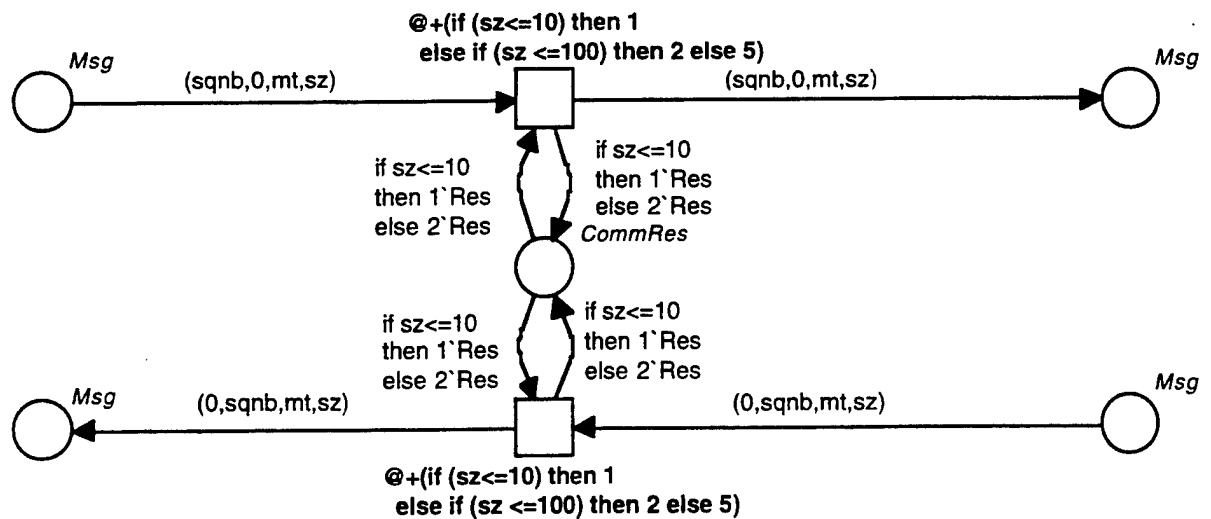


Figure 7 Model of the Communications Network

Model of the Squadron

The generic model of the squadron is shown in Figure 8. This model is instantiated four times to represent the four squadrons of the Air Combat Operations System. The data specific to each instance are defined with the token contained in the input port place of color set "SquadData." The first data of color set "Squad" is the ID number and is used for the reception and the generation of messages. The second data is the coordinate of the wing, where the images should be downloaded and is used for the formulation of requests.

The squadron receives a message when the transition "Receive Message" fires. Only the messages addressed to a particular squadron enables the transition of this particular page/class instance. The messages received by a squadron can be of two types: either a mission, or an image. When a mission message is received, the transition "Prepare mission" fires enabling the transition "Assess need for ext. data." Each squadron has a local image database by images sent by the wing as a response to a query of the wing image database or to a request for satellite observation. The mission preparation can be completed only if recent images are present in the image database, that is if the time stamp of the image is larger than 0. The firing of the transition "Assess need for ext. data" can result in different markings. If there are recent images in the local database, no token is generated and the mission preparation can be completed. If there is no image in the local database then the wing image database has to be queried first: the firing of the transition "Assess need for ext. data" results in a token being generated in the image of color set "Image" input to the transition "Query Wing Image DB." The firing of this transition results in the creation of a query message which is sent when the transition "Send Message" fires.

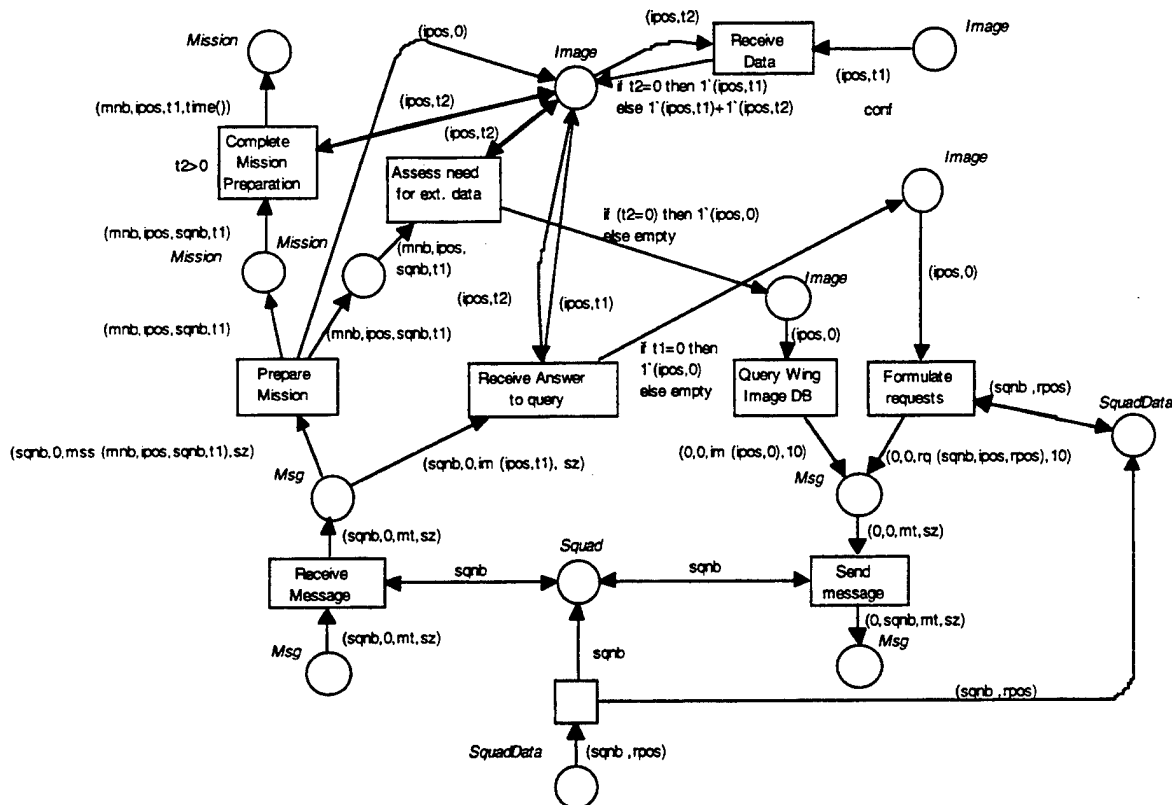


Figure 8 Model of the Squadron

As mentioned earlier, image messages from the wing are received when tokens appear in the input port place of color set "Msg." These image messages are either responses to query to the wing database or the response to requests for satellite observation. The firing of the transition "Receive Message" generates a token in the place of color set "Msg" that enables the transition "Receive Answer to query." If the image received is recent, the local image database is updated and the transition "Complete Mission Preparation" can be enabled and fires. In the image is not recent, a request for satellite observation needs to be formulated and the firing of this transition "Receive answer to query" results in a token being generated in the place of color set "Image" input to the transition "Formulate requests."

Model of the Ground Control Station

The model of the Ground Control Station is shown on Figure 9. Requests are received by the Ground Control stations when a token appears in the input port place of color set "SortReq," enabling the transition "Process Request." The firing of this transition results in a "SatCmd" token being generated, one for each satellite. This satellite command is sent when the satellite is in transmission range of the ground control station. This is implemented by the guard function the transitions "Send command" which can fire only when the difference of the latitudes and longitudes of the satellites and ground control station position are less than 20° . There is a transition "Send Command" for each satellite. The model is constructed so that for each request, only one satellite command is sent to each satellite. The model does not predict the trajectories of the satellites to infer which ground control station will be able to send first the commands to the satellite.

Model of the Satellite

The generic model page of the satellite is presented in Figure 10. The model page is instantiated twice to consider the two satellites necessary for the full observation of the Earth in less than 24 hours. The data specific to each satellite is specified by the initial marking of the page of color set "Position" at the top right side of the page and that forms a self loop only with the transition "Move." The place of color set "Position" on the left of this transition contains the current position of the satellite. The transition "Move" is enabled every unit of time when the token "ut" contained in the place of color set "UT" becomes available. When the transition "Move" fires, the old position of the satellite is withdrawn from the left place of color set "Position" and is

replaced by the new position computed on the output arc expression “newposit(time(),fi,teta)” which computes the position at the current time (function time(), and uses the parameters fi and teta of the trajectory obtained from the right place of color set “Position.” fi is the position on the orbital plane at time $t = 0$, teta is the longitude of the intersection of the orbital plane and the equatorial plane at time $t = 0$. The firing of this transition results also in the time stamp of the token “ut” increased by 1, so that the transition can be enabled when the simulation time advances.

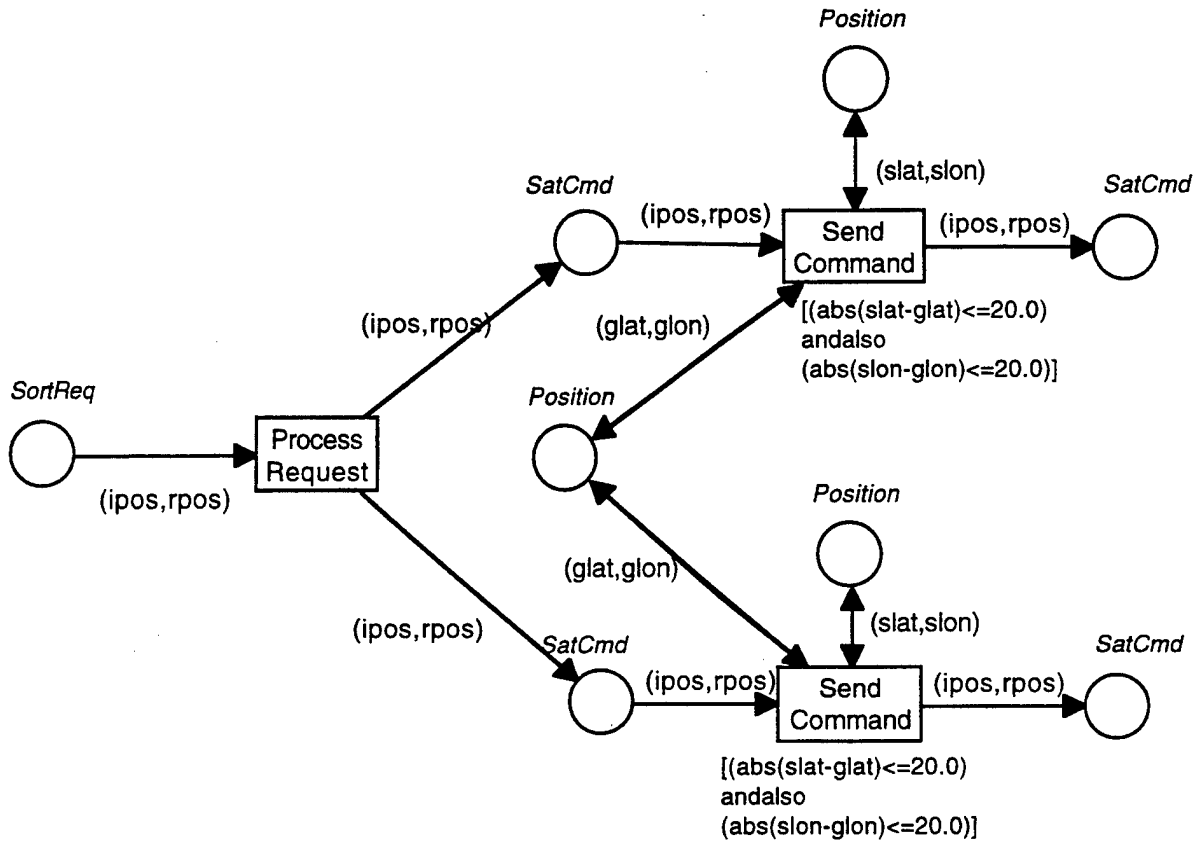


Figure 9 Model of the Ground Control Station

The satellite command sent by one of the ground control stations is received by the satellite when a token of type “SatCmd” appears in the input port place of color set “SatCmd,” enabling the transition “Receive Request.” The command specifies two things: the location of the center of the image to be taken as represented by the variable ipos or (ilat, ilon), and the location of the Eagle Vision where the image should be downloaded. The picture is taken only when the satellite is in observation range of the desired coordinates as implemented by the guard function of the transition “Take Picture.” The transition “Take Picture” fires only when the difference of the longitudes and latitudes of the satellite and the area of interest positions is less than 5.1° . When the transition fires,

a token image is generated in the place of color set "Image" with a time stamp equal to the current time of the simulation and the SatCmd token is reproduced in the input place "SatCmd" to the transition "Send Picture." This transition is enabled when the guard function indicating that the difference of the longitudes and latitudes of the satellite and Eagle Vision positions is less than 20° evaluates to true. This guard function models the fact that the satellite needs to be in transmission range of the Eagle Vision to download the data. When the transition "Send Picture" fires, the token Image is generated in the output port place of color set "Image."

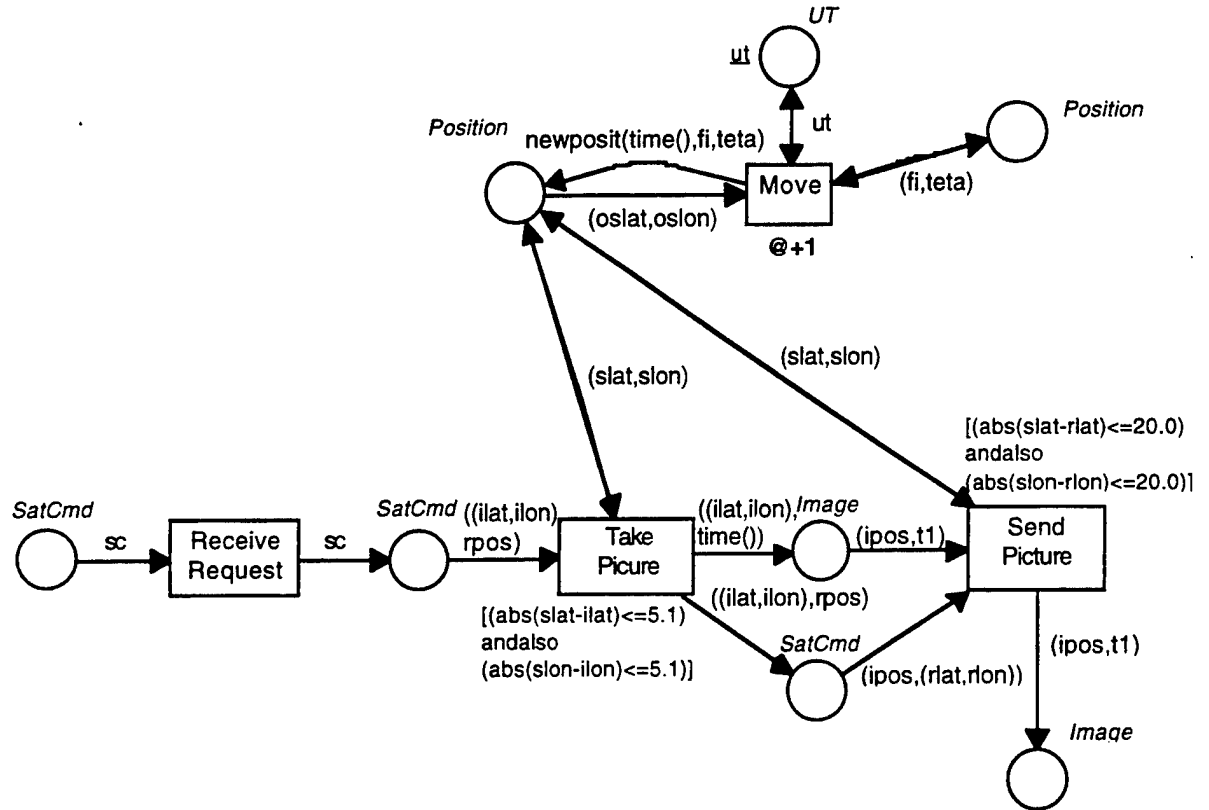


Figure 10 Model of the Satellite

2.5 Conclusion

The current model includes the reception and distribution of the ATO, the process of mission preparation and the transmission of external information. The model is deemed sufficient to serve as a testbed for the results of the other tasks.

3. ADAPTATION OF RESULTS FROM TEMPORAL LOGIC

3.1. Introduction

This section presents an approach for the modeling and subsequent analysis of the temporal aspects of discrete-event systems. A system's temporal aspects might be represented in terms of properties that hold for certain time intervals, processes taking some time to complete, and/or events/occurrences requiring virtually no time to take place. In order to model a system's temporal aspects one needs, therefore, to have a formalism capable of handling both interval and point descriptions of various components of the system.

An extension of Allen's interval logic (Allen, 1981a, 1981b, 1983, 1984; Allen and Hayes, 1985a, 1985b; Galton, 1990) that incorporates both descriptions of time, point and interval, is presented. The new formalism, called *point-interval logic*, extends the axiomatic system of interval logic by adding new axioms to it. A Petri net model is shown to model this axiomatic system of the point-interval logic. This Petri net based approach transforms the system's specifications given by temporal statements of point-interval logic into Petri net structures. A temporal inference engine (TIE) based on this Petri net representation infers new temporal relations among system intervals, identifies temporal ambiguities and errors (if present) in the system's specifications, and finally identifies the windows of capabilities defined by the user. *The real strength of this inference engine is that it performs all these functions in linear time, completely avoiding the combinatorial nature of the problem inherent in the inference engines of its kind.*

The presentation is organized as follows. Section 3.2 presents the definitions of different notions and terms used in the approach together with an introduction to point-interval logic. The section also presents an analytical model for the temporal relations. A transformation of this model to an equivalent Petri net representation is presented in Section 4. The structure of Petri net representing temporal relations among intervals is shown to reveal inconsistencies and incompleteness present in the system. The graph-based inference engine, TIE, is presented in section 4.3. Section 4.4 contains the results of the approach and the algorithm. Section 4.5 applies the result of the methodology to a simple problem, while Section 4.6 describes TEMPER1, the software implementation of the methodology.

3.2. Mathematical Model

Definition 1: *Interval*

An interval is denoted by a symbol, i.e., a letter or a digit, and is defined as

$X = [sx, ex]$, where $ex \geq sx$.

In this definition sx denotes 'start of x ' and ex 'end of x '. The two bounds are abstract notions and may not have numerical values assigned to them. The intervals considered in this paper are all closed intervals.

Definition 2: *Interval Length*

The length of an interval $X = [sx, ex]$, denoted by $|X|$, is defined as $|X| = ex - sx$

Definition 3: *Point*

An interval X with $|X| = 0$ is a point interval; $x = [px, px]$ is a point interval or point.

In a system description a point may be used to signify an occurrence or an event.

Definition 4: *Sup-interval and Sub-interval*

Let $X = [sx, ex]$ and $Y = [sy, ey]$ are two intervals. The sup-interval ' $X \bullet Y$ ' is defined as $[sx, ey]$ only if $ex = sy$ — the two intervals are consecutive intervals.

Similarly an interval $X = [sx, ex]$ can be represented in terms of a series of consecutive intervals, called sub-intervals of X ; $X = x_1 \bullet x_2 \bullet \dots \bullet x_n$, where $x_1 = [sx, a_1]$, $x_2 = [a_1, a_2]$, ..., $x_n = [a_{n-1}, ex]$ and $|x_i| \geq 0$.

Definition 5: *Length of Sup-interval*

Let $X \bullet Y$ be a sup-interval with sub-intervals X and Y , the length of the interval $|X \bullet Y|$ is given as $|X \bullet Y| = |X| + |Y|$

Definition 6: *Temporal Relation R_i*

The temporal relation R_i is a truth functional binary relation defined as:

$R_i: I \times I \rightarrow \{T, F\}$

where I is the set of intervals and, T and F are the Boolean values, true and false

Definition 7: *Temporal Relation R_i*

The temporal relation R_i is a binary relation defined as: $R_i: I \times I \rightarrow I^*$

where I is the set of intervals; and I^* is the set of intervals, sup-intervals, and sub-intervals formed by the intervals in I .

The relation $X R_i Y$ holds if and only if $X R_i Y$ is true.

Definition 8: Composite Interval

Let $X = [sx, ex]$ and $Y = [sy, ey]$ are two intervals, where $sx = sy$ and $ex = ey$, then they are jointly represented by a composite interval $[X; Y] = [sx;sy, ex;ey]$. By definition $[X, X] = [X]$ or simply X .

Definition 9: Composite Point

Let $X = [px]$ and $Y = [py]$ represent a single point on the time line, i.e., $px = py$, then the composite point is represented as $[X;Y] = [px;py]$.

Point-interval Logic

The point-interval logic presented in this section is an extension of Allen's interval logic. The formalism considers a single time line. Any two intervals with non-zero lengths on this time line are related by one of the seven relationships presented in Figure 11, case I. The underlying assumption of the interval logic is the non-zero lengths of time intervals. The present approach relaxes this assumption and allows intervals with zero lengths — points. Case II, Figure 11, presents two possible temporal relations between two points. Finally, case III, Figure 11, presents the possible set of temporal relations between a point and an interval. From now on, the term interval is used to refer both intervals and points if not explicitly stated otherwise.

Definition 10: Set of Temporal Relations, R

R represents the set of temporal relations R_i and is given as:

$R = \{\text{Before, Meets, Overlaps, Starts, During, Finishes, Equals}\}$

Proposition 1

The temporal relations presented in Figure 11 are mutually exclusive and exhaustive, i.e.,

- 1) if $(X R_i Y)$, $R_i \in R$, then there does not exist an $R_j \in R$, such that $(X R_j Y)$ holds true.
- 2) for any two intervals X and Y there must exist an $R_i \in R$ such that either $(X R_i Y)$ or $(Y R_i X)$ holds true
(with the exception of Equals relation where $(X \text{ Equals } Y) = (Y \text{ Equals } X)$.)

Properties of Ri

The relationship that exists between two intervals is independent of the actual lengths of the individual intervals. However, some of the relations in R do imply a measure between the relative lengths of the intervals being compared. This information about the relative lengths of intervals can be exploited by a specification and analysis system, based on temporal logic, to elicit the implicit temporal information present in a system of temporal statements, i.e., length of an interval X can be calculated with the help of known temporal relations among other intervals in the systems and X, and with the known lengths of some of these intervals. Similarly, the information about the relative lengths of intervals can be used to identify inconsistencies and errors present in the system specifications. However, the methodology does not take into consideration either the actual or relative lengths of intervals present in a system. Based on the definition of temporal relations shown in Figure 11, Figure 12 presents the measure of relative lengths of two intervals.

Other properties of the individual relations Ri are:

1. **Before**

- a) Irreflexive, i.e., X **Before** X.
- b) Antisymmetric, i.e., either X **Before** Y or Y **Before** X is true but not both.
- c) Transitive, i.e., X **Before** Y Y **Before** Z \rightarrow X **Before** Z.

2. **Meets**

- a) Irreflexive
- b) Antisymmetric
- c) Nontransitive

3. **Overlaps**

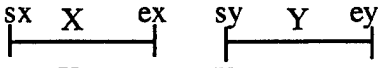
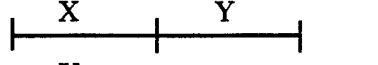
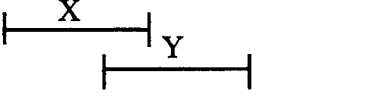
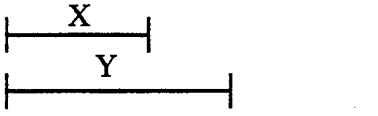
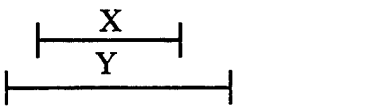
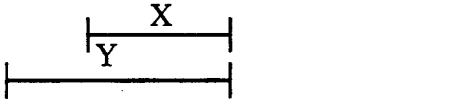
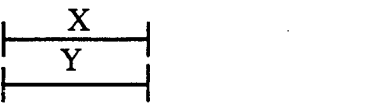
- a) Irreflexive
- b) Antisymmetric
- c) Transitive

4. **Starts**

- a) Irreflexive
- b) Antisymmetric
- c) Transitive

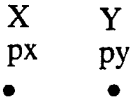
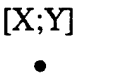
CASE I— X and Y both intervals with non-zero lengths:

$X = [sx, ex]$, $Y = [sy, ey]$ with $sx < ex$ and $sy < ey$

- | | | | |
|----|---------------------|-------------------------------------|---|
| 1. | X Before Y | $ex < sy$ |  |
| 2. | X Meets Y | $ex = sy$ |  |
| 3. | X Overlaps Y | $sx < sy$
$sy < ex$
$ex < ey$ |  |
| 4. | X Starts Y | $sx = sy$
$ex < ey$ |  |
| 5. | X During Y | $sx > sy$
$ex < ey$ |  |
| 6. | X Finishes Y | $sy < sx$
$ey = ex$ |  |
| 7. | X Equals Y | $sx = sy$
$ex = ey$ |  |

CASE II—X and Y both points:

$X = [px]$ and $Y = [py]$ with $sx = ex = px$ and $sy = ey = py$

- | | | | |
|----|-------------------|-----------|--|
| 1. | X Before Y | $px < py$ |  |
| 2. | X Equals Y | $px = py$ |  |

CASE III— X is a point and Y is an interval:

$X = [px]$ and $Y = [sy, ey]$ with $sx = ex = px$ and $sy < ey$


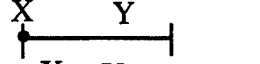
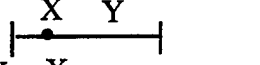
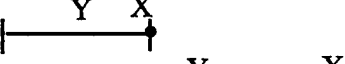
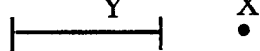
- | | | | |
|----|---------------------|----------------|---|
| 1. | X Before Y | $px < sy$ |  |
| 2. | X Starts Y | $px = sy$ |  |
| 3. | X During Y | $sx < px < ey$ |  |
| 4. | X Finishes Y | $px = ey$ |  |
| 5. | Y Before X | $ey < px$ |  |

Figure 11 Temporal Relations

X Starts Y	$ X < Y $
X During Y	$ X < Y $
X Finishes Y	$ X < Y $
X Equals Y	$ X = Y $

Figure 12 Measure of Relative Length of Two Intervals

5. During

- a) Irreflexive
- b) Antisymmetric
- c) Transitive

6. Finishes

- a) Irreflexive
- b) Antisymmetric
- c) Transitive

7. Equals

- a) Reflexive
- b) Symmetric
- c) Transitive

Analytical Model

An analytical representation of temporal relations among intervals is shown in Figure 11. According to this representation a temporal relation between two intervals can be described as a set of algebraic inequalities among points representing start and end of these intervals. Figure 13 presents a general description of this model.

Two points, p_1 and p_2 , on a single time line can be related to each other by one of the following four relations:

- | | |
|-------------------|------------------|
| (i) $p_1 > p_2$ | (ii) $p_1 = p_2$ |
| (iii) $p_1 < p_2$ | (iv) $p_1 ? p_2$ |

(? represents 'unknown'. This is added to incorporate incomplete information)

$X = [sx, ex], Y = [sy, ey]$ with $sx \leq ex$ and $sy \leq ey$	
<i>X Before Y</i>	$sx \leq ex < sy \leq ey$
<i>X Meets Y</i>	$sx < ex = sy < ey$
<i>X Overlaps Y</i>	$sx < sy < ex < ey$
<i>X Starts Y</i>	$sx = sy \leq ex < ey$
<i>X During Y</i>	$sy < sx \leq ex < ey$
<i>X Finishes Y</i>	$sy < sx \leq ex = ey$
<i>X Equals Y</i>	$sx = sy \leq ex = ey$

Figure 13 Algebraic Description of Temporal Relations between Intervals

With these four possibilities and the analytical model of Figure 13, a temporal relation Ri between two intervals X and Y , denoted as $Ri(X, Y)$ can be represented as a 4-digit string made of elements from the alphabet $\{<, =, >, ?\}$, where the first (left-most) digit represents the relation between sx and sy , second digit between sx and ey , third digit represents relation between ex and sy , and fourth between ex and ey . The representation does not take into account the point or interval nature of X and Y . The information about the nature of intervals is assumed to be known; however, this 4-digit representation can be extended to accommodate this feature as well. Figure 14 shows this string representation for each temporal relation.

The description of a temporal relation Ri (known or unknown or partially known) between two intervals X and Y , therefore, require only 8 bits (for a computer implementation.) This byte representation of a temporal relation between two intervals, and the nature of the intervals (which requires 1 bit for each interval) are all that is required to store (and maintain) complete, or partially/totally incomplete information about the two intervals. Even if one needs to store temporal relations between all possible combinations of intervals from a set of n intervals, it will take $n + n(n - 1)/2$ bytes to store all this information. Temporal inconsistencies can be identified by checking inconsistent patterns of this 8-bit representation.

$R_i(X, Y)$	$sx \ Vs \ sy$	$sx \ Vs \ ey$	$ex \ Vs \ sy$	$ex \ Vs \ ey$
X Before Y	<	<	<	<
X Meets Y	<	<	=	<
X Overlaps Y	<	<	>	<
X Starts Y	=	<	>(or =)	<
X During Y	>	<	>	<
X Finishes Y	>	<(or =)	>	=
X Equals Y	=	<(or =)	>(or =)	=
X unknown Y	?	?	?	?

Figure 14 String Representation of Temporal Relations

Axioms of Point-Interval Logic

Point Axioms

Let p_1 , p_2 , and p_3 be points defined on a single time line. The following four (1-4) axioms can be used to infer the temporal relation between two points on the time line. The remaining combinations of temporal relation among three points (L.H.S. of the axioms) result in an unknown, ?, relation between the two points at the R.H.S. of the axioms; this is described by the axiom 5, where underscore, $_$, is used to denote any of the four temporal relation, $\{<, =, >, ?\}$.

1. $(p_1 < p_2) \wedge (p_3 < p_1) \rightarrow (p_3 < p_2)$
2. $(p_1 < p_2) \wedge (p_3 = p_1) \rightarrow (p_3 < p_2)$
3. $(p_1 < p_2) \wedge (p_3 = p_2) \rightarrow (p_3 > p_1)$
4. $(p_1 = p_2) \wedge (p_3 = p_1) \rightarrow (p_3 = p_2)$
5. $(p_1 _ p_2) \wedge (p_3 _ p_1) \rightarrow (p_3 ? p_2)$

Together with the 4-digit representation of a temporal relation and the nature of the intervals involved, point axioms implement the axiomatic system of the point-interval logic. The following examples illustrate some of the axioms of point-interval logic.

Example 1

Let X, Y and Z be three intervals, with following known relations among them:

(i) **X Before Y**

(ii) **Y Before Z**

The corresponding string representations are given as follows:

	$s_x V s_y$	$s_x V e_y$	$e_x V s_y$	$e_x V e_y$
X Before Y	<	<	<	<
	$s_y V s_z$	$s_y V e_z$	$e_y V s_z$	$e_y V e_z$
Y Before Z	<	<	<	<

The application of point axioms to these two strings yields:

$s_x V s_z$	$s_x V e_z$	$e_x V s_z$	$e_x V e_z$	
<	<	<	<	X Before Z

The example implements the following axiom of point-interval logic:

$$(X \text{ Before } Y) \wedge (Y \text{ Before } Z) \rightarrow (X \text{ Before } Z)$$

Example 2

Let X, Y and Z be three intervals, with following known relations among them:

(i) **X Overlaps Y**

(ii) **Y Overlaps Z**

The corresponding string representations are given as follows:

	$s_x V s_y$	$s_x V e_y$	$e_x V s_y$	$e_x V e_y$
X Overlaps Y	<	<	>	<
	$s_y V s_z$	$s_y V e_z$	$e_y V s_z$	$e_y V e_z$
Y Overlaps Z	<	<	>	<

The application of point axioms to these two strings yields:

$$\begin{array}{cccc}
sx & Vs & sz & sx & Vs & ez & ex & Vs & sz & ex & Vs & ez \\
< & & & < & & & ? & & & < & &
\end{array}$$

The resulting string corresponds to one of the following relations between X and Z:

- (a) **X Before Z** (obtained by replacing '?' by '<')
- (b) **X Meets Z** (obtained by replacing '?' by '=')
- (c) **X Overlaps Z** (obtained by replacing '?' by '>')

The corresponding axiom of the point-interval logic can now be described as:

$$(X \text{ Overlaps } Y) \wedge (Y \text{ Overlaps } Z) \rightarrow (X \text{ Before } Z) \vee (X \text{ Meets } Z) \vee (X \text{ Overlaps } Z)$$

The logical operator ' \vee ' is defined as follows:

$$A_1 \vee A_2 \vee \dots \vee A_n = (A_1 \hat{A}_2 \dots \hat{A}_n) \vee (\hat{A}_1 A_2 \dots \hat{A}_n) \vee (\hat{A}_1 \hat{A}_2 \dots A_{n-1} \hat{A}_n) \vee (\hat{A}_1 \hat{A}_2 \dots \hat{A}_{n-1} A_n)$$

where \hat{A} is Not (A).

Inference Engine

One may visualize an inference engine based on the axiomatic system presented in the previous section. Suppose we have a known temporal relation between two intervals X and Y. Now, if a new temporal relation between two intervals Y and Z is discovered (or supplied by the user), the inference engine constructs an analytical representation of the temporal relation between X and Z with the help of known relations among X, Y and Z, and the point axioms presented in the previous section. The resulting string representation of the relation between X and Z is pattern matched with the string representations of Figure 14 to infer possible temporal relation(s) between X and Z. However, the construction of the analytical representation of an unknown relation between two intervals, X and Z, from the known temporal relation among system's other intervals can be accomplished via a combinatorially large number of ways. The following example illustrates this issue:

Example 3

Let a system be described in terms of the following statements:

- 1) **W Meets X**
- 2) **X Meets Y**
- 3) **W Before U**
- 4) **Y Finishes U**

The inference engine is required to infer temporal relation between intervals W and Y. In the example statements, there are two possible alternatives to calculate the unknown relation; a) with the help of known relations (**W Meets X**), and (**X Meets Y**), infer the temporal relation between W and Y; b) with the help of known relations (**W Before U**), and (**Y Finishes U**), infer the temporal relation between W and Y. Both alternatives imply the same temporal relation between the two intervals: (**W Before Y**).

The example illustrates the fact that an inference engine based on the axiomatic system of the point-interval logic has to search all possible combinations of the known temporal statements in order to explore the right combination(s) of statements that yields the result. The inference required to calculate an unknown relation may not be a single step inference (as illustrated in the example, where two statements are used to infer a third one.) Consider a situation where an unknown relation between X and Y is required to be inferred, and the known statements are: (X R0 I1), (I1 R1 I2), ..., (Ii Ri Ij), ..., (In Rn Y). In such a situation, a chain of inference is required to derive the unknown relation between X and Y. The search for all such combinations of statements that (may) yield the required result among all possible combinations of known statements makes the inference process computationally intensive. (The combinations that may yield the required result are referred to as feasible combinations.) One may suggest that an inference engine does not have to search for all feasible combinations of statements; the search can be halted as soon as the first such combination is encountered, and a required relation is inferred. An inference engine with this approach can only be applied to a system of temporal statements which is known to be *consistent* apriori. A formal definition of consistency in temporal logic follows in the next section. The following example illustrates the issue:

Example 4

Let a system be described in terms of the following statements:

- 1) **W Meets X**
- 2) **X Meets Y**

3) **Y Meets Z**

4) **Z Meets W**

The inference engine is required to infer temporal relation between intervals W and Y. One may infer (W **Before** Y) with the help of statements 1, and 2. However, if statements 3 and 4 are considered, the inference mechanism infers (Y **Before** W), which is in contradiction to the previous inference. Both inferences are correct so far as the inference mechanism is considered, however, the error is caused by the inconsistency present in the input statements.

An inference engine for the point-interval logic, therefore, requires an exhaustive enumeration of the result through all feasible combinations of statements, provided no knowledge of system's correctness is available apriori. An inference engine that outputs the result as soon as it finds the first feasible set of inputs can only be employed to a known consistent system of temporal statements. This, in turn, requires a front-end verification mechanism for the inference engine that verifies the system's correctness prior to applying the axioms of the point-interval logic. A detailed discussion and solution to this problem is presented in the next section.

4. TEMPORAL LOGIC / PETRI NET METHODOLOGY: TEMPER1

4.1 Single Time Line - Single Future (SLSF)

The complete formal axiomatic system presented in Section 3.2 for the point-interval logic can now be used to model temporal aspects of a system. This section presents a graph based approach, termed Temporal Logic/Petri Net (TL/PN) formalism, for the implementation of the axiomatic system of point-interval logic. The TL/PN approach transforms the system's specifications given by temporal statements into a graph structure. The graph-based temporal inference engine (TIE) identifies temporal ambiguities and errors (if present) in the system's specifications, infers new temporal relations among system's intervals, and finally identifies the windows of capabilities defined by the user. The temporal inference engine, TIE, of the TL/PN methodology performs all these tasks by completely avoiding the combinatorial nature of the inference mechanism discussed in the previous section.

In the present approach all input statements are connected together with an implicit conjunction; they are all valid simultaneously. Therefore, they represent a system's temporal description on a single time line: for any point on this time line there will exist only one future.

Petri Net Representation

The analytical model of a temporal relation R_i presented in Figure 13 is transformed to a Petri Net (PN) representation by the following approach:

- A point t_1 is represented as a transition, labeled $[t_1]$.
- Two points t_1, t_2 with $t_1 = t_2$ are represented as a single transition, labeled $[t_1; t_2]$.
- Two points t_1, t_2 with $t_1 < t_2$ are represented as two transition, $[t_1]$ and $[t_2]$, with a *link* from $[t_1]$ to $[t_2]$.

Definition 10: *Link*

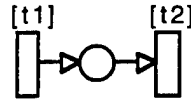
A place P is said to constitute a link from $[t_i]$ to another $[t_j]$ if the following holds:

$$*P = \{[t_i]\} \text{ and } P^* = \{[t_j]\}$$

where $*P$ and P^* represent the pre- and post-set of the place P .

Example 5

Let $t1$ and $t2$ be two points on a single time line with $t1 < t2$, the Petri net representation of the temporal relation between the two points is shown as:



For the sake of simplicity the PN is transformed to a simpler graph representation with each transition in the PN represented as a node (vertex) and a link between two transitions as an arc (edge) between the two nodes representing the transitions. The temporal relation of the example is now represented as:



The example presents a much simpler graphical representation for the temporal relation between two time points. However, the insistence on using the Petri net formalism for modeling temporal statements is due to a future extension of TL/PN methodology capable of incorporating alternative inputs. A discussion on this issue is delayed for a later section. From now on, instead of using the Petri net representation, the simple graph representation is used for illustration purposes. The new graph representation is termed as Point Graph (PG) representation.

Definition 11: *Point Graph*

A point graph $PG(V, E)$ is a directed graph with each node or vertex $v \in V$ representing a point on a time line and each edge $e_{12} \in E$, between two vertices $v1$ and $v2$, representing a temporal relation ' $<$ ' between the two vertices — ($v1 < v2$).

A temporal relation Ri between two intervals X and Y can now be represented by an equivalent Point graph representation, as shown is Figure 15.

The TL/PN approach takes statements in the point-interval logic and transforms each individual statement into an equivalent PG representation. The PG representing the entire system of temporal statement is then constructed by unifying individual PGs to a (possibly) one connected graph. The unifying process employs a very simple approach which merges two nodes into a single node if the labels of the two nodes contains at least one point in common. The following definition presents a formal description of this unification process.

CASE I— X and Y both intervals with non-zero lengths:

$X = [sx, ex]$, $Y = [sy, ey]$ with $sx < ex$ and $sy < ey$

1. *X Before Y* $[sx] \rightarrow [ex] \rightarrow [sy] \rightarrow [ey]$
2. *X Meets Y* $[sx] \rightarrow [ex;sy] \rightarrow [ey]$
3. *X Overlaps Y* $[sx] \rightarrow [sy] \rightarrow [ex] \rightarrow [ey]$
4. *X Starts Y* $[sx;sy] \rightarrow [ex] \rightarrow [ey]$
5. *X During Y* $[sy] \rightarrow [sx] \rightarrow [ex] \rightarrow [ey]$
6. *X Finishes Y* $[sy] \rightarrow [sx] \rightarrow [ex;ey]$
7. *X Equals Y* $[sx;sy] \rightarrow [ex;ey]$

CASE II—X and Y both points:

$X = [px]$ and $Y = [py]$ with $sx = ex = px$ and $sy = ey = py$

1. *X Before Y* $[px] \rightarrow [py]$
2. *X Equals Y* $[px;py]$

CASE III— X is a point and Y is an interval:

$X = [px]$ and $Y = [sy, ey]$ with $sx = ex = px$ and $sy < ey$

1. *X Before Y* $[px] \rightarrow [sy] \rightarrow [ey]$
2. *X Starts Y* $[px;sy] \rightarrow [ey]$
3. *X During Y* $[sy] \rightarrow [px] \rightarrow [ey]$
4. *X Finishes Y* $[sy] \rightarrow [px;ey]$
5. *Y Before X* $[sy] \rightarrow [ey] \rightarrow [px]$

Figure 15 Point Graph Representation of Temporal Relations

Definition 12: Unification

Let $v1 = [p_i; \dots; p_n]$ and $v2 = [p_j; \dots; p_m]$ be two nodes in a PG representation. If there exist a point p_k such that $p_k \in [p_i; \dots; p_n]$ and $p_k \in [p_j; \dots; p_m]$, then the two nodes are merged into a single node $v12$ such that:

$$v12 = [p_i; \dots; p_n] \cup [p_j; \dots; p_m]$$

$$*v12 = *v1 \cup *v2$$

$$v12^* = v1^* \cup v2^*$$

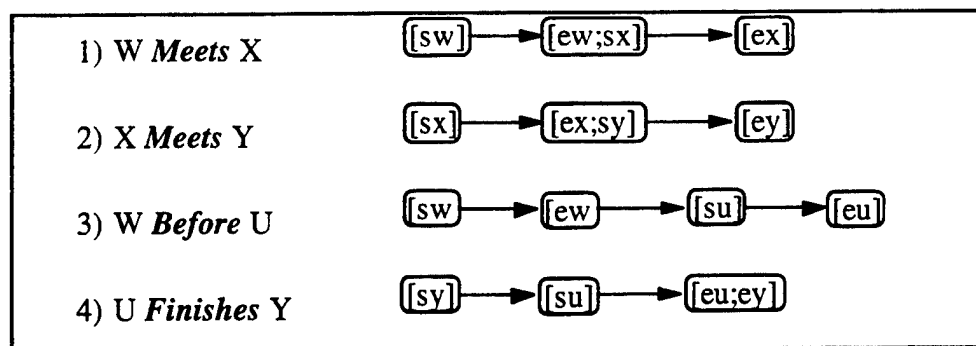
The following example illustrates the unification process.

Example 6

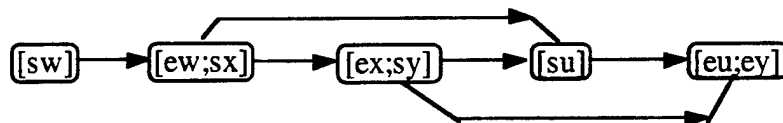
Let a system be described in terms of the following statements:

- 1) **W Meets X**
- 2) **X Meets Y**
- 3) **W Before U**
- 4) **Y Finishes U**

The point graph representation of individual statements is given as:



Unifying these individual statements yields:

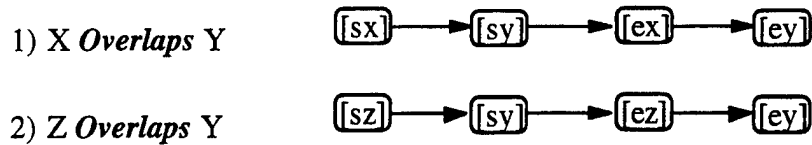


Example 7

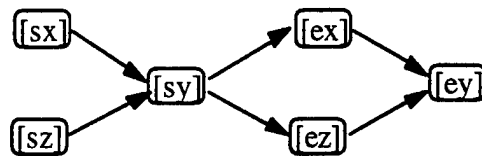
Let a system be described in terms of the following statements:

- 1) **X Overlaps Y**
- 2) **Z Overlaps Y**

The PGs representing these statements are:



After unifying the individual nets, the unified net looks like:



Definition 13: Ordering

A relation R on a set A is an ordering, if and only if

- R is reflexive
- R is antisymmetric
- R is transitive

Definition 14: Total Ordering (Chain)

An ordering R on a set A is a total ordering or chain if and only if given any $(x, y) \in A^2$, either $x R y$ or $y R x$.

If an ordering is not a total ordering, it is called partial ordering.

Definition 15: Cover

By y covers x is meant that $x R y$ and that there is no element z , $z \neq x, y$ such that $x R z R y$.

Definition 16: Connected Chains

A chain $x_0 < x_1 < \dots < x_n$ will be connected if x_i covers x_{i-1} for all i .

Proposition 2

The set of nodes, V , in a unified PG is ordered (possibly partially) by the relation ' $<$ ' (depicted as arcs in PG representation.)

A path between two nodes in a PG may not be a connected chain as is the case in Example 7. However, identification of connected chains in a PG may be a desirable feature for a computer implementation of this methodology. Filtering out all the chains that are non-connected may result in a smaller graph with no loss of information. This issue is left for a future treatment of this subject.

4.2 System Verification

The PG representation of a system's temporal aspects organizes the information contained in temporal statements into a graphical structure. This section presents an analysis of this graphical representation. The analysis applies certain graph-theoretic concepts on the structure of point graphs, identifies their structural properties, and finally interprets the results obtained in terms of the temporal aspects of the systems under consideration.

As mentioned in Section 3.2, the inference engine of the point-interval logic requires a consistent system specification in order to infer temporal relation among system intervals. A verification methodology is presented in this section, which makes use of the structural properties of the PG to detect inconsistencies, if present in the system description.

Inconsistency in Temporal Logic

Definition 17: *Inconsistency (Allen, 1981)*

A set of statements (inferences) is said to be inconsistent if they can not all be true at the same time.

Definition 18: *Inconsistency in Point-interval Logic*

A system's description contains inconsistent information if for some intervals X and Y both $X R_i Y$ and $X R_j Y$, $i \neq j$, or $X R_i Y$ and $Y R_j X$ (with the exception of $R_i = R_j = \text{Equal}$) hold true.

In the point representation of intervals, the inconsistency is defined as follows:

Definition 19: Inconsistency in Point-interval Logic

A system's description contains inconsistent information if for some points t_1 and t_2 (representing start and/or end of intervals) the following hold true:

$$t_1 < t_2 \text{ and } t_1 = t_2 \quad \text{or} \quad t_1 < t_2 \text{ and } t_1 > t_2$$

Proposition 3

A set of temporal statements is inconsistent if and only if the PG representation of the set contains *self-loops* and/or cycles.

Proof

Proof of the Proposition follows from the definition of inconsistency (Definition 18) and the construction of PGs.

Definition 20: Self-loop

In a Petri net, a place is said to constitute a self-loop if it is both an input and output place of the same transition.

In a point graph an arc forms a self-loop if it originates and ends in the same node.

A consistent set of temporal statements is, therefore, characterized by Proposition 4:

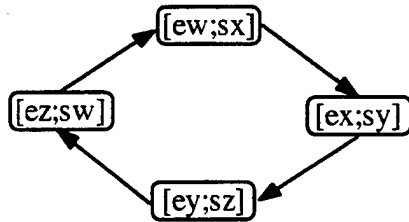
Proposition 4

A set of temporal statements is consistent if and only if the PG representation of the set is an acyclical graph.

The following examples illustrate the results presented in Proposition 3.

Example 8

The PG representing the statements in Example 4 is shown as follows; the cyclic structure of the PG reveals the inconsistency present in the system.



Example 9

For a system's description given as follows, the corresponding unified PG will be:

- 1) W Meets X
- 2) X Meets Y
- 3) W Meets Y



The presence of self-loop identifies an inconsistency present in the system.

The self-loops present in a PG representation can be easily identified by individually scanning all the nodes in the graph. The discussion so far has characterized inconsistency in a set of temporal statements and has illustrated the ways in which an inconsistent set of statements reveals itself in the PG representation. The following are some graph-theoretic results that help calculate the cycles inside a point graph.

Proposition 5

The underlying Petri net of a *connected* point graph is a *Marked* graph.

Proof

The proof follows from the definition of Marked graphs and the construction of point graphs.

Definition 21: *Marked Graph*

A Marked graph is a connected Petri net in which each place has exactly one input and one output transition.

Proposition 5 allows the application of following results on PG representation.

Theorem 1 (Hillion, 1986)

The *S-components* of *minimal support S-invariants* of a marked graph are exactly its directed elementary circuits. (Note: For the definitions of S-component, and minimal support, refer to Zaidi (1992).)

Definition 22: *S-Invariant*

Given an incidence matrix C of a Petri net, an S-invariant is a $n \times 1$ non-negative integer vector X of the kernel of C^T , i.e., $C^T X = 0$

Definition 23: *Connectivity Matrix* (Zaidi and Levis, 1995)

A point graph with n directed arcs and m nodes can be represented by a $n \times m$ matrix J , the *Connectivity matrix*. The rows correspond to arcs, the columns correspond to nodes.

- $J_{ij} = 1$ if the directed arc in i -th row originates from the j -th node.
- $J_{ij} = -1$ if the directed arc in i -th row terminates in the j -th node.
- $J_{ij} = 0$ if the directed arc in i -th row is not connected to j -th node.

Proposition 6

The connectivity matrix of a PG is exactly the incidence matrix of its underlying Petri net.

Proof

Proof follows from the definitions of connectivity and incidence matrices, and the construction of a point graph from a Petri net.

Now, using results from Proposition 6 and Theorem 1, the cycles present in a PG can be identified as follows:

Proposition 7

A point graph contains cycles if and only if it has non-zero S-invariants calculated with the help of connectivity matrix.

Once cycles are detected in a PG by calculating non-zero S-invariants, the nodes responsible for these cycles can be easily identified. This will, in turn, identify intervals involved in these cycles. This information can be used to correct the system of temporal statements.

Incompleteness

The PG model constructed for a system also helps identify the missing temporal relation among system intervals, thus provides means for elicitation of a *complete* specification of system's temporal aspects. A complete system specification is characterized by the following definition.

Definition 24: *Completeness*

A system's description is complete if for all intervals X and Y , there exists a temporal relation $R \in \mathbf{R}$ which is either provided explicitly or can be inferred through the axiomatic system.

The application of Definition 24 on the PG representation of a system intervals gives rise to the following definition of completeness.

Definition 25: Completeness in the PG Representation

A system's description is complete if for all nodes X and Y in its PG representation, there exists a relation '<' between them.

Proposition 8

A system's description is complete if and only if the PG representing the system is an acyclical structure with:

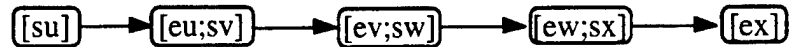
- a) exactly one source node and one sink node, and
- b) one connected chain from the source node to the sink node that contains all the nodes in the system description.

Example 10

Let a system be described in terms of the following statements:

- 1) U Meets V
- 2) V Meets W
- 3) W Meets X

The PG representing the system is shown as:



According to Definition 25 and Proposition 8, the system specification is complete. A matrix establishing a temporal relation between any pair of two intervals of the system is shown below:

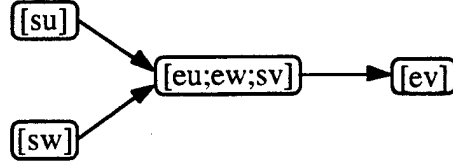
	U	V	W	X
U	Equals	Meets	Before	Before
V	—	Equals	Meets	Before
W	—	—	Equals	Meets
X	—	—	—	Equals

Example 11

Let a system be described in terms of the following statements:

- 1) U Meets V
- 2) W Meets V

The PG representing the system is shown as:



The PG represents an incomplete system specification since no temporal relation can be established between intervals U and W from the given input.

Although the PG in Example 11 does not establish a temporal relation between intervals U and W, the application of axioms on this representation yields that the only temporal relations that can hold between the two interval are 'Equals' and 'Finishes'— a fact also apparent in the PG representation because both U and W have the same end points. Introduction of any other relation between the two intervals will inevitably introduce loops in the PG representation, thus forcing inconsistencies in the system.

The TL/PN formalism, therefore, can be used to identify incompleteness and then to elicit a complete specification of a system's temporal aspects by filling in the missing relations in a consistent manner.

Redundancy

Redundancy in a system of temporal statements refers to the presence of multiple copies of the same relation.

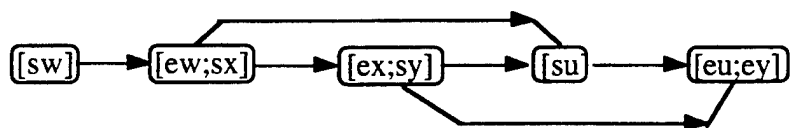
Definition 26: *Redundancy*

A system's description contains redundant information if one or all of the following cases are present:

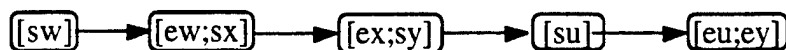
- a) Multiple copies of a temporal relation, $X Ri Y$, between two intervals,
- b) $X Ri Y$ both explicitly and implicitly (can be inferred by axioms) given.

Some of the redundant cases, as characterized by the definition, are automatically removed in the PG representation of the system; relations appearing as multiple arcs between two adjacent nodes are filtered out in the construction of PG representation. A relation between two nodes x and y in PG representation, represented by an arc from x to y , also introduces a redundancy if y does not cover x . Example 6 presents such redundant cases. Figure 16a reproduces the PG in Example 6. An equivalent representation of the PG in Figure 16a is shown in 16b, where the redundant relations among nodes are filtered out.

The redundant cases illustrated in Figure 16 increase the size of the PG representation, and therefore require more storage than an equivalent PG representation without these cases. On the other hand, from a computational point of view, the presence of such cases may not directly affect the system performance in an adverse manner. However, an approach, although expensive in terms of computational requirements, is presented that can be used to filter out such redundancies.



(a) PG of Example 6



(b) PG without redundant arcs

Figure 16 Redundancy in PG representation

The approach first constructs a virtual external node, P_0 , in the PG representation and draws arcs from it to all external node. A connectivity matrix J is then constructed for this new PG. The S-invariants are calculated the source nodes in the PG. Similarly, arcs are drawn from all the sink nodes in the PG to this for J . The calculated S-invariants correspond to all the directed paths from source nodes of the PG to sink nodes (assuming there are no internal loops in the PG representation.) The maximal T-supports of the calculated S-invariant correspond to all the connected chains in the PG. A point graph formed by these connected chains (excluding the external node) would yield an equivalent representation of the PG with out redundant arcs.

Definition 26: *T-support*

If X is an S-invariant, the set of input and output nodes of the arcs whose corresponding components in X are strictly positive is the T-support of the invariant, noted $\langle X_T \rangle$.

The T-support of an S-invariant is said to be maximal if and only if it is not contained in the T-support of another S-invariant but itself.

4.3 The TL/PN Inference Engine (TIE)

Once a consistent (error free) description of a system is derived and represented in terms of a PG structure, the calculation of unknown temporal relation among system intervals and windows of capabilities becomes a mere search problem in the net. The advantage of TL/PN formalism is that it not only verifies system correctness prior to any inference making, but also overcomes the combinatorial problem, discussed in Section 3.2, associated with inferring new temporal relations.

Definition 27: Window of Capability

A window of capability associated with intervals X_1, X_2, \dots, X_n is defined to be a composite interval $[x_1; x_2; \dots; x_n]$ where x_i is a sub-interval of X_i for $i = 1, 2, \dots, n$.

The TL/PN inference engine (TIE) infers temporal relation between two intervals $X (= [sx, ex])$ and $Y (= [sy, ey])$ by constructing the string representation (Figure 14) of the temporal relation between the two intervals by searching for the directed paths between the nodes representing the intervals in the PG representation and using the facts that ' $sx \leq ex$ ' and ' $sy \leq ey$ '. The search for the directed paths between two nodes in a PG requires the use of two algorithms called *FPSO* and *FPSI* (Zaidi and Levis, 1997), two variants of an earlier FindPath algorithm (jin, 1986). The *FPSO*(p) algorithm, when applied to a node p in a PG, collects all the nodes that have directed paths to p . The *FPSI*(p) algorithm, on the other hand, collects all the nodes to which p has a directed path. The existing implementation of the two algorithms uses a depth-first search strategy to calculate the result. The search terminates as soon as it encounters the desired node x during its search or encounters all the sink nodes without hitting the desired node. Tables 1 and 2 present the formal descriptions of the two techniques. The notation $x \rightarrow p$, used in the tables, represents the binary relation that a directed path exists from node x to node p .

Table 1 FindPath-to-Sources (FPSO) Algorithm

For a PG = (V, E)
 $p \in V$
 $FPSO(p) = S$
 where
 $S = \{x \mid x \in V, x \twoheadrightarrow p\}$
 $\forall x \in S, x < p$

Table 2 FindPath-to-Sinks (FPSI) Algorithm

For a PG = (V, E)
 $p \in V$
 $FPSI(p) = S$
 where
 $S = \{x \mid x \in V, p \twoheadrightarrow x\}$
 $\forall x \in S, p < x$

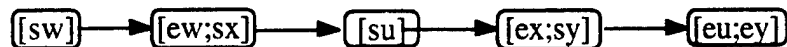
The following examples illustrate the mechanism of TIE in calculating the unknown temporal relation between intervals and windows of capabilities associated with system intervals.

Example 12

Consider the system of temporal statement introduce in Example 3:

- 1) W Meets X
- 2) X Meets Y
- 3) W Before U
- 4) Y Finishes U

The PG representing the set of temporal statements is:



The inference engine is then asked to calculate the temporal relation between the intervals W and Y, represented as “?-R(W, Y)” in TIE formalism. The inference engine tries to build the

string representation for $R(W, Y)$ by first searching for the relation between points 'ew' and 'sy'. Since $sy \in FPSI(ew)$, TIE calculates 'ew < sy'. Using the fact that 'sw < ew' and 'sy < ey', the entire string representation is calculated to be:

$$\begin{array}{cccc} sw & Vs & sy & sw & Vs & ey & ew & Vs & sy & ew & Vs & ey \\ < & & < & & < & & < & \end{array}$$

The string corresponds to '**W Before Y**'— returned by TIE.

Table 3 presents example queries and their return values as implemented in the inference engine TIE for the calculation of temporal relation between intervals.

Example 13

Consider the temporal system in Example 12. Let the window of capability associated with intervals U and X is desired to be calculated, denoted as '?-window(U, X)'. The inference engine processes this query by the following algorithm:

```
?-window(U, X)
IF (?-U Overlaps X) THEN return [sx, eu]
ELSE IF (?-X Overlaps U) THEN return [su, ex]
ELSE IF (?-U Starts X) THEN return [su;sx, eu]
ELSE IF (?-X Starts U) THEN return [sx;su, ex]
ELSE IF (?-U During X) THEN return [su, eu]
ELSE IF (?-X During U) THEN return [sx, ex]
ELSE IF (?-U Finishes X) THEN return [su, eu;ex]
ELSE IF (?-X Finishes U) THEN return [sx, eu;ex]
ELSE IF (?-U Equals X) THEN return [su;sx, eu;ex]
ELSE return []
```

For the example system the inference engine will return [su, ex] as the window of capability associated with intervals U and X.

Note that this algorithm should include the definition:

with $?-window([], X_i) = []$ (where [] corresponds to 'no' in TIE formalism)

Table 3 Queries for Calculating Temporal Relations

Example Queries	Return Value	Comments
?-R(X, Y)	X Ri Y where Ri's $\in R \cup \{\text{unknown}\}$	The inference engine returns a specific Ri if it finds one, otherwise it outputs all possible relations that might exist without creating an inconsistency, given incomplete information available.
?-X Ri Y	Yes No Plausible	The inference engine returns 'yes' if the input statement can be inferred by TIE, 'no' if the inferred statement is in contradiction to the input, and 'plausible' if the input statement is one of the possibilities that can exist in the set of temporal statements without introducing inconsistencies.
?-R([p1, p2], [p3, p4])	X Ri Y where Ri's $\in R \cup \{\text{unknown}\}$	This query presents a generalized format of the first query. This syntax helps a user in constructing custom intervals by selecting arbitrary points in the PG as start and end of these user-defined intervals, with 'p1 ≤ p2' and 'p3 ≤ p4'.
?-[p1,p2] Ri [p3, p4]	Yes No Plausible	This query presents a generalized format of the second query. Using this syntax a user can construct intervals by selecting arbitrary points in the PG as start and end of these user-defined intervals, with 'p1 ≤ p2' and 'p3 ≤ p4'.

A recursive definition for the calculation of window of capability associated with n intervals follows:

$$?-window(X_1, X_2, \dots, X_n) = ?-window(?-window(X_1, X_2, \dots, X_{n-1}), X_n)$$

Table 4 presents the general form of queries and their return values as implemented in the inference engine TIE for the calculation of windows of capabilities.

Table 4 Queries for Calculating Windows of Capabilities

Example Queries	Return Value	Comments
$?-window(X_1, X_2, \dots, X_n)$	$[p_i, p_j] \in I^*$ or $[]$ (no) if none exists	The inference engine returns an interval if it finds one, otherwise it outputs $[]$ or no.
$?-window([p_1, p_2], \dots, [p_{n-1}, p_n])$	$[p_i, p_j] \in I^*$ or $[]$ (no) if none exists	This query presents a generalized format of the first query. Using this syntax a user can construct intervals by selecting arbitrary points in the PG as start and end of these user-defined intervals, with ' $p_1 \leq p_2$ ' and ' $p_{n-1} \leq p_n$ '.

4.4 The Algorithm

The TL/PN methodology can be organized in the steps listed below:

Assumption: System has only one time line with a single future (SLSF).

Step 1: Input statements in point-interval logic (i.e., X Before Y.)

Step 2: Construct a unified Point Graph.

Step 3: Verify system correctness by applying the S-invariant algorithm to the connectivity matrix of the PG. Report the inconsistent cases and halt.

Step 4: Invoke Q&A (Question and Answer) system of TIE for calculating temporal relation and windows of capabilities.

An overall view of the TL/PN methodology is presented in Figure 17. The shaded box labeled as 'Language Translator' has not been addressed yet.. The current approach requires the user to interact directly with the system using the strict syntax of point-interval logic.

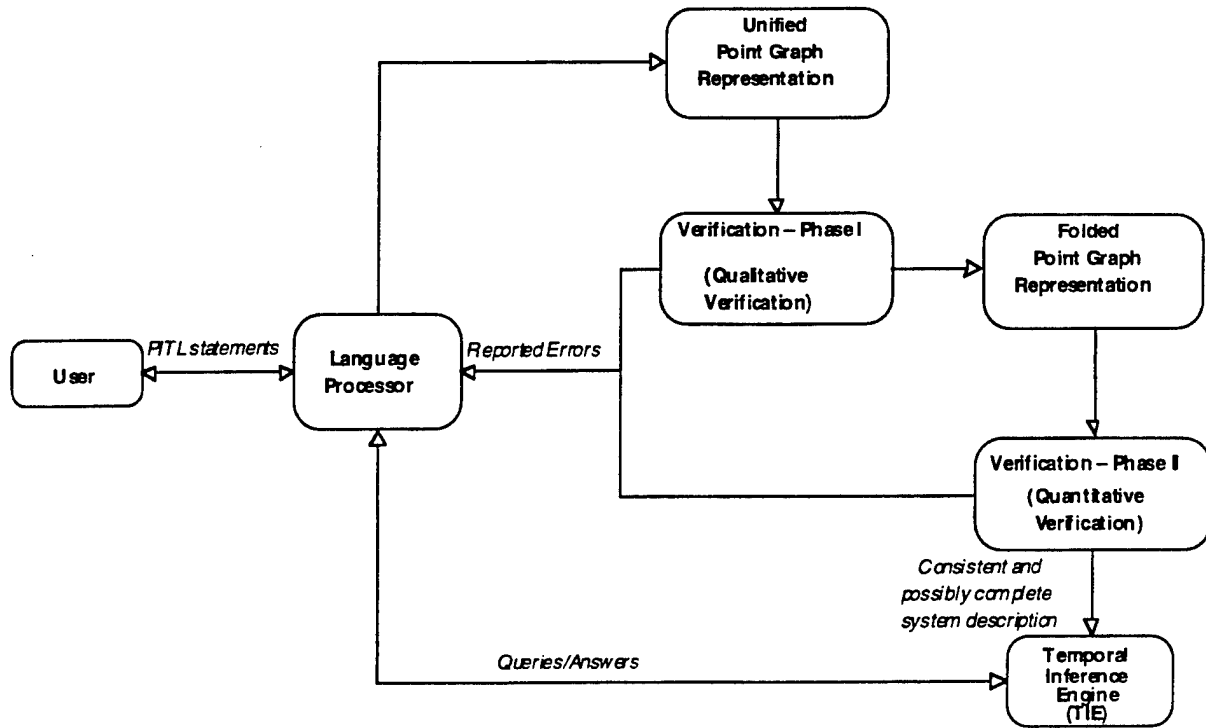


Figure 17 An Overall View of the Methodology

The syntax of point-interval logic requires statements to be given in a very strict and unintuitive form. Every two intervals in a system must be described by one of the 13 mutually exclusive temporal. Once a system's description is provided in this tight syntactical form, a PG model can be constructed by the approach presented in an earlier section. It was felt that the syntactical requirements of the input to such a methodology are highly unintuitive and require a lot to be done by the user before a PG model is built and analyzed for correctness and calculations of windows of capabilities. A front-end user interface, therefore, requires a 'Language Translator' that takes English-like statements as input and extracts information about intervals and the temporal relationships among them in terms of statements of interval logic. For this purpose a context-based grammar is required. The following example illustrates the issue.

Example 14

Let a system has the following two statements given in a descriptive form:

(1) Resource R1 allocated to P at 6:00/day1.

(2) Resource R1 allocated to Q at 7:00/day1.

If a translator is given these statements in the sequence shown above, the translator, on receiving the first statement, extracts the following temporal statements with accompanying semantics associated with each interval symbol used:

(1) Resource R1 allocated to P at 6:00/day1.

Temporal Statements	Semantics
X Starts Y	Interval Symbol: X Interval Semantics: Allocated(R1, P) Type: Event Time: 6:00/day1 Interval Symbol: Y Interval Semantics: Holds(R1, P) Type: Property Start Time: 6:00/day1 End Time: unknown

Similarly on receiving statement 2:

Temporal Statements	Semantics
W Finishes Y	Interval Symbol: W Interval Semantics: Released(R1, P) Type: Event Time: 7:00/day1 <i>Interval Symbol: Y</i> <i>Interval Semantics: Holds(R1, P)</i> <i>Type: Property</i> <i>Start Time: 6:00/day1</i> <i>End Time: 7:00/day1</i>
W Starts Z	<i>Interval Symbol: W</i> Interval Semantics: Allocated(R1, Q) Type: Event Time: 7:00/day1 Interval Symbol: Z Interval Semantics: Holds(R1, Q) Type: Property Start Time: 7:00/day1 End Time: Unknown

This example illustrates a number of issues, mainly:

Context:

The context of the statements introduces a number of additional temporal relations among intervals, which are not explicit in the input statements. For example, the statement 'W **Finishes** Y' can not be extracted if the context of interval Y is unknown—which comes from the first statement. The order in which these statements are input also affects the way these output statements are extracted. The only way out of this problem is a context-based grammar that can translate input statements into statements in point-interval logic.

User Interaction:

In the example, it is assumed that the resource R1 is a unique single entity which can be assigned to Q only if taken from P first. If this were not the case the output of such a system would not contain 'W **Finishes** Y'. It is, therefore, required that such a translator could resolve these ambiguities with human interaction. A translator would require explicit information about perishable, multiple, etc. resources to understand the context associated with intervals.

The issue of a 'Language Translator', as a part of TL/PN methodology, is left for a future treatment of the subject. The rationale for discussing it here is to provide the reader an overall view of the methodology and the issues involved.

4.5 A Simple Example

Suppose a system's description is given as follows:

- 1) Resource 1 allocated to P at 06:00/Day 1.
- 2) Resource 1 freed by Q at 11:00/Day 1.
- 3) Resource 1 moved from P to Q at 09:00/Day 1.
- 4) Resource 2 moved from Q to P at 07:00/Day 1.
- 5) Resource 2 freed by P at 10:00/Day 1.

Step 1:

The temporal aspects of the system are formally represented in terms of statements of point-interval logic given as follows:

1) (X Starts Y)

where $X = [6 \ 6]$ is a point specifying the event of allocating Resource 1 to P, and Y is the interval during which P is in possession of Resource 1.

2) (U Finishes Z)

where $U = [11 \ 11]$ is a point specifying the event of Resource 1 being taken from Q, and Z is the interval during which Q is in Possession of Resource 1.

3) (W Finishes Y) (W Starts Z)

where $W = [9 \ 9]$ is a point specifying the event of reallocation of Resource 1 from P to Q, and Y and Z are the intervals specified before.

4) (V Finishes K) (V Starts R)

where $V = [7 \ 7]$ is a point specifying the event of reallocation of Resource 2 from Q to P, K is the interval during which Q is in possession of Resource 2, and R is the interval during which P is in possession of Resource 2.

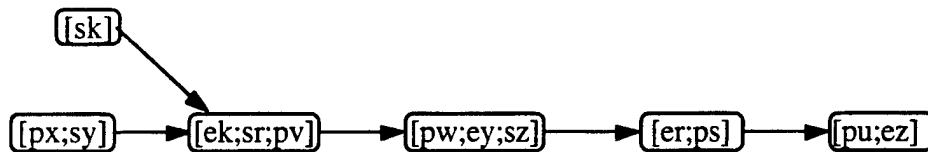
5) (S Finishes R)

where $S = [10 \ 10]$ is an empty interval specifying the event of Resource 2 being taken from P, and R is the interval during which P is in Possession of Resource 2.

6) (X Before V) (V Before W) (W Before S) (S Before U)

Step 2:

The point graph constructed by unifying the PGs obtained for individual statements is shown below.



Step 3:

The net has no self loops or cycles, therefore, has no temporal inconsistency. Note that the system description is incomplete since the node [sk] does not have a directed path to/from the node labeled as [px;sy] in the PG representation of the system.

Step 4:

Now if it is required to know the precise period (interval) during which Resources 1 and 2 are both in the possession of P— window of capability associated with intervals Y and R— the inference engine is asked the following query:

?-window(Y, R)

Since (?-Y **Overlaps** R) returns a value 'yes', the found window of capability is: [sr, ey] or [ek;sr;pv, pw;ey;sz] in the composite representation.

4.6 TEMPER 1: Software Implementation of the Algorithm

The algorithm presented in Section 4.5 has been implemented in the framework of Design/CPN as a suite of programs written in Meta Language (ML). ML is a functional program developed in Edinburgh which is imbedded in the editor/simulator Design/CPN. This suite is called TEMPER 1 (TEMPoral programmER, version 1)

LEXICAL STRUCTURE

Key Words (case sensitive): Before, Meets, Overlaps, Starts, During, Finishes, Equals.

User-defined Identifiers:

Points: Any string of alphanumeric characters starting with a lower-case alphabetic character.

<point> —> <lower-case-letter><letter|digit>*

Examples: x, y, point, point1, event1, etc.

Intervals: Any string of alphanumeric characters starting with an upper-case alphabetic character.

<interval> —> <upper-case-letter><letter|digit>*

Examples: X, Y, Interval, Process1, etc.

SEMANTIC/SYNTACTIC STRUCTURE OF INPUT STATEMENTS

- <interval> <R> <interval>

where <R> —> Before| Meets| Overlaps| Starts| During| Finishes| Equals

Example: Process1 Before Process2

- $\langle \text{point} \rangle \langle R1 \rangle \langle \text{interval} \rangle$
where $\langle R1 \rangle \rightarrow$ Before| Starts| During| Finishes
Example: event1 Starts Process2. The statement 'event2 Overlaps Process1' is semantically incorrect, since a point (event1) can not 'Overlaps' an interval (Process1).
- $\langle \text{interval} \rangle \langle R2 \rangle \langle \text{point} \rangle \emptyset$
where $\langle R2 \rangle \rightarrow$ Before
Example: Process1 Before event2.
- $\langle \text{point} \rangle \langle R3 \rangle \langle \text{point} \rangle \emptyset$
where $\langle R3 \rangle \rightarrow$ Before| Equals
Example: event1 Before event2

SEMANTIC/SYNTACTIC STRUCTURE OF QUERIES

- $?- D([\langle \text{slelp} \rangle \langle \text{interval} \rangle, \langle \text{slelp} \rangle \langle \text{interval} \rangle], [\langle \text{slelp} \rangle \langle \text{interval} \rangle, \langle \text{slelp} \rangle \langle \text{interval} \rangle])$

Example: $?- D([sProcess1, eProcess2], [pevent1, pevent1])$

Note: The point representation of intervals used in the queries provides user to construct user-defined intervals by specifying any start and end points, i.e., $[sX, sY]$ is an interval with starting point sX (start of some other interval X) and ending point sY (end of some other interval Y). Similarly, $[px, py]$ be an interval constructed with the help of two existing points in the system. The only condition to be satisfied is that the end point should be greater than the start point. If a user specify an interval with the help of two arbitrary points not satisfying the condition, the TEMPER1 query processor would generate an error "Inconsistent interval specification". A point, on the other hand, is represented as $[px, px]$, $[sX, sX]$, or $[eX, eX]$.

A SAMPLE RUN OF TEMPER1: CASE 1: WORKING ON A NEW EXAMPLE

- Invoke "Design/CPN™".
- Open TEMPER1. A page with the diagram shown in Figure 18 appears.
- Import or write input temporal statements in a text box.

Input Statements

x Starts Y
u Finishes Z
w Finishes Y
w Starts Z
v Finishes K
v Starts R
s Finishes R
x Before v
v Before w
w Before s
s Before u

- Select box labeled "Start Up1", and type "command -" (or invoke Start ML from Aux menu).
- Select box labeled "Start Up2", and type "command -".

This will load all the required functions in the memory.

From now on, selecting a box and invoking "command -;" will be referred as "Execute" the box.

- Execute box labeled "Input". The box will start flashing. Take the cursor on top of the text box (any) with input temporal statements, the box will start flashing, click on it. The language processor will run a syntax/syntactic/semantic check and report errors (if any).
- If no errors are found in the input, Execute box labeled "Draw PGs".

This will draw point graphs for each input statements. (Figure 19)

- Execute box labeled "Self Loops?". If self loops are reported, correct them and restart.
- If no self loops are found, Execute "Unify PGs". If this box is executed in the presence of errors, the processor will automatically remove the self loops. This will construct a unified PG representing the input set of temporal statements (See Figure 20).
- Generally the Point Graph generated by TEMPER1 will need to be cleaned up. Use the cursor to select and rearrange the Point Graph as shown in Figure 21. Design/CPN provides several "Align" functions that can assist in this process.

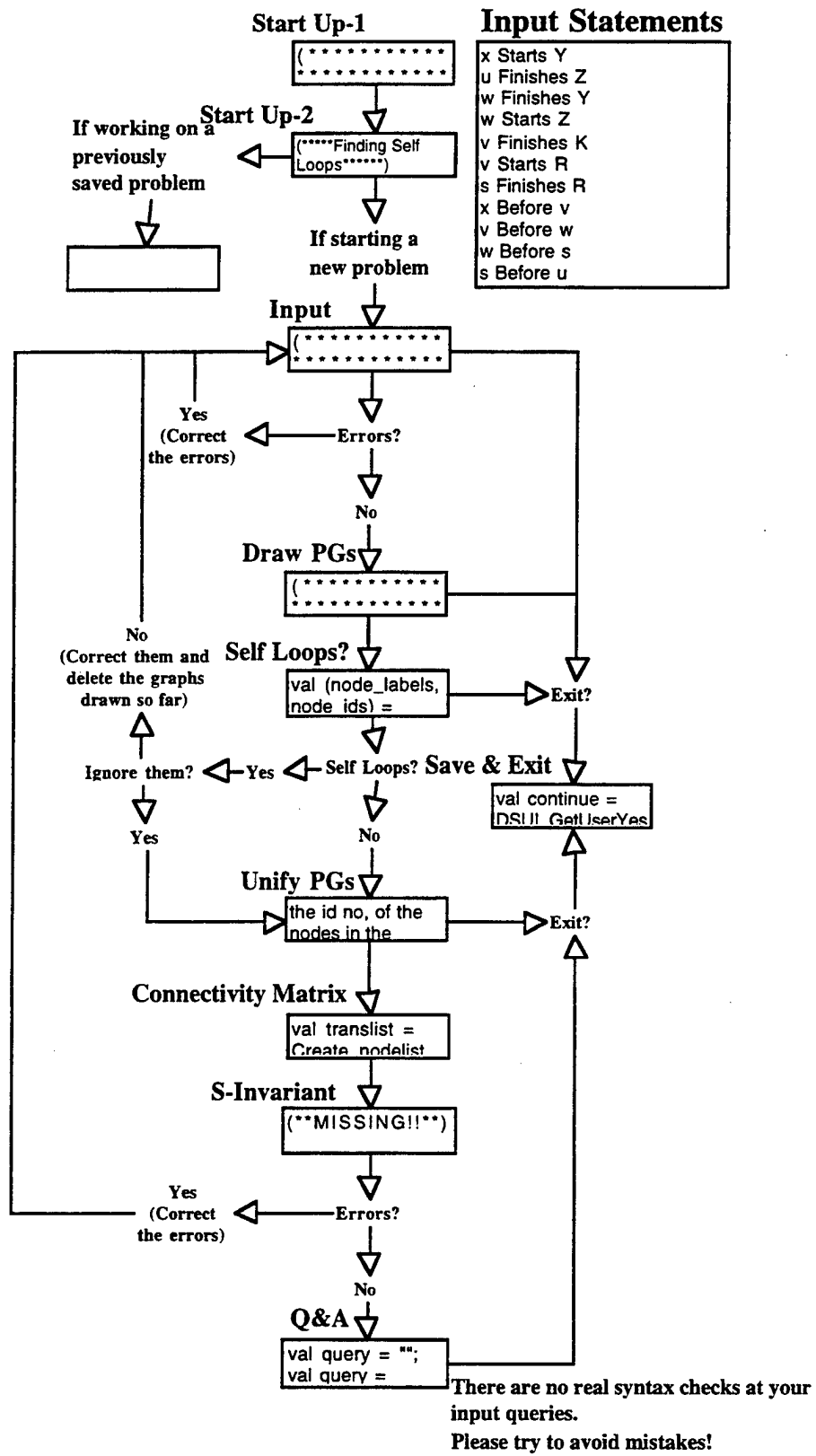


Figure 18 TEMPER1 Graphical User Interface

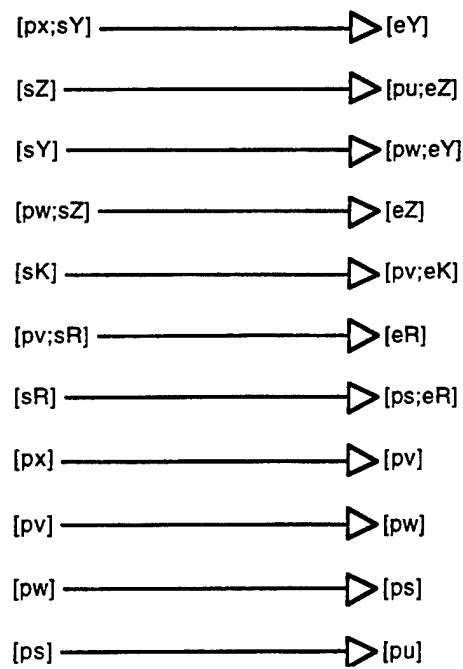


Figure 19 Point Graph Generated by TEMPER1

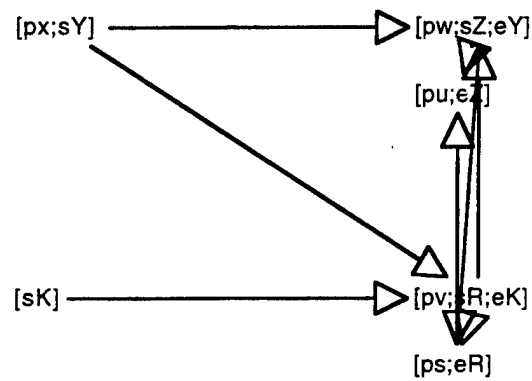


Figure 20 Unified Point Graph Generated by TEMPER1

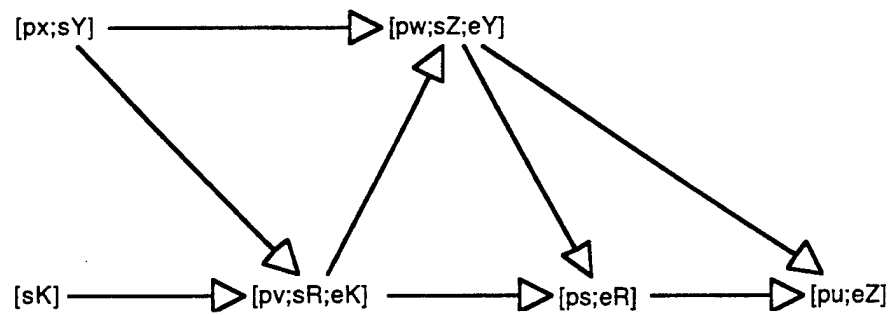


Figure 21. Re-Arranged Unified Point Graph

- Execute box labeled "Connectivity Matrix". A text box will appear on the screen with the connectivity matrix (or incidence matrix) of the unified PG (Figure22).
- The box labeled "S-invariants" does not contain any code and can not be executed. The text box containing the connectivity matrix can be exported out as text and be used as input to the S-invariant algorithm developed by the System Architectures Laboratory. The reported S-invariants identify errors in the input, which should be corrected and the processing restarted.

number of places: 8 number of transitions: 6						
Trans=	[px;sY]	[pw;sZ;eY]	[pu;eZ]	[sK]	[pv;sR;eK]	[ps;eR]
[px;sY]<[pw;sZ;eY]	1	-1	0	0	0	0
[pw;sZ;eY]<[pu;eZ]	0	1	-1	0	0	0
[sK]<[pv;sR;eK]	0	0	0	1	-1	0
[pv;sR;eK]<[ps;eR]	0	0	0	0	1	-1
[px;sY]<[pv;sR;eK]	1	0	0	0	-1	0
[pv;sR;eK]<[pw;sZ;eY]	0	-1	0	0	1	0
[pw;sZ;eY]<[ps;eR]	0	1	0	0	0	-1
[ps;eR]<[pu;eZ]	0	0	-1	0	0	1

Figure 22 Connectivity Matrix

- If no errors are found, Execute box "Q&A". A dialog box will appear with a default query syntax in it (Figure 23).

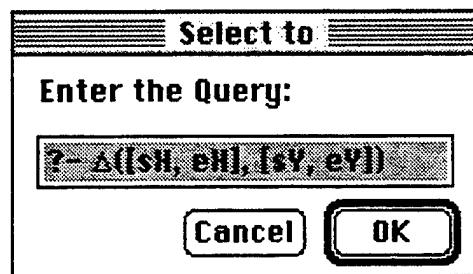


Figure 23 "Q&A" Box

- Insert your query and click OK. If TEMPER1 detects an error in the query, the following error message will appear (Figure 24). In this example, there is no Interval “X”.

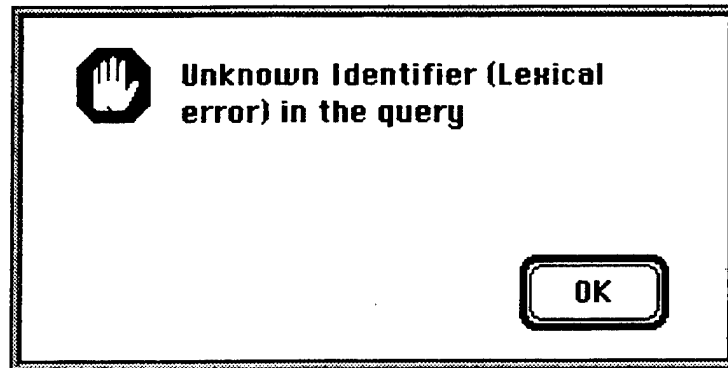


Figure 24 “Q&A” Error Message

- Click OK and re-insert the correct query as shown in Figure 25.

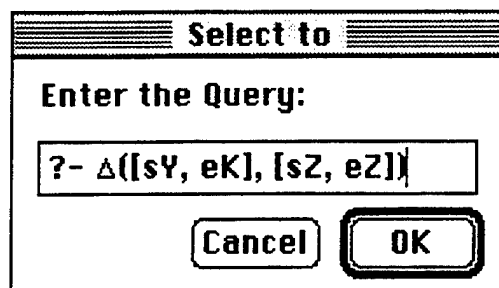


Figure 25 Correct Query Entry

TEMPER1 responds with the result of the query as shown in Figure 26.

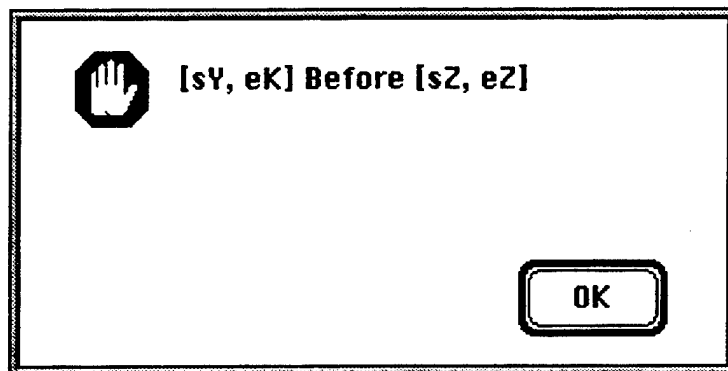


Figure 26 TEMPER1 Response to Query

- Execute "Q&A" every time a query is required to be processed.

- Execute box "Save&Exit" before quitting (even if you are quitting in the middle of your problem solving). TEMPER1 will provide the window shown in Figure 27. If you click "Yes" TEMPER1 will save all parameters needed to continue working on the problem at a later time.

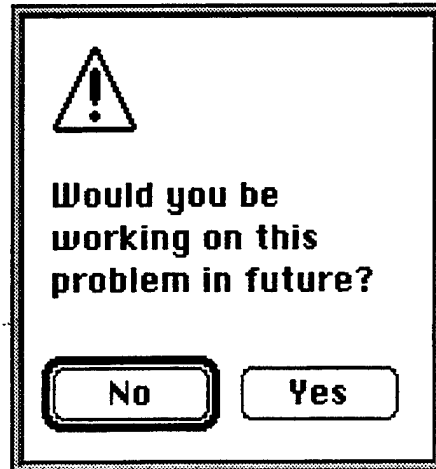


Figure 27. TEMPER1 Window When Save&Exit Is Executed

- After Selecting "Yes", TEMPER1 provides the reminder window shown in Figure 28. Click OK and quit Design/CPN.

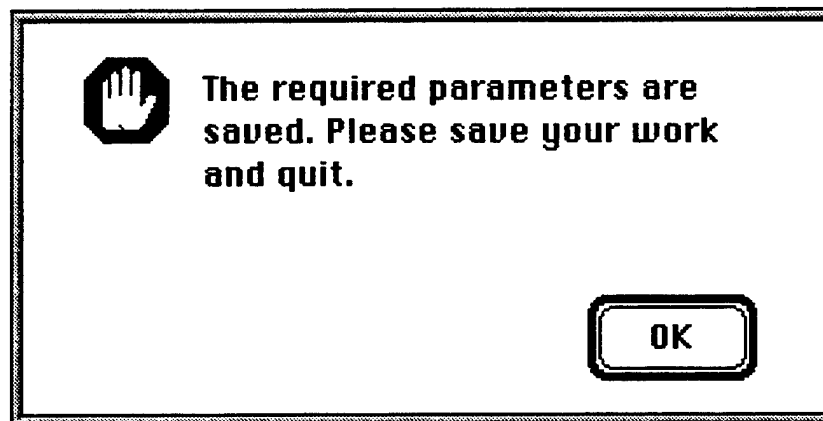


Figure 28. TEMPER Reminder Window Prior To Quitting

Case 2: WORKING ON A PREVIOUSLY SAVED EXAMPLE

- Execute "Start UP1" and then "Start UP2".

- Execute box to the left of "Start Up2" and follow the instructions. TEMPER1 provides a window indicating where you stopped working on the problem (Figure 29).

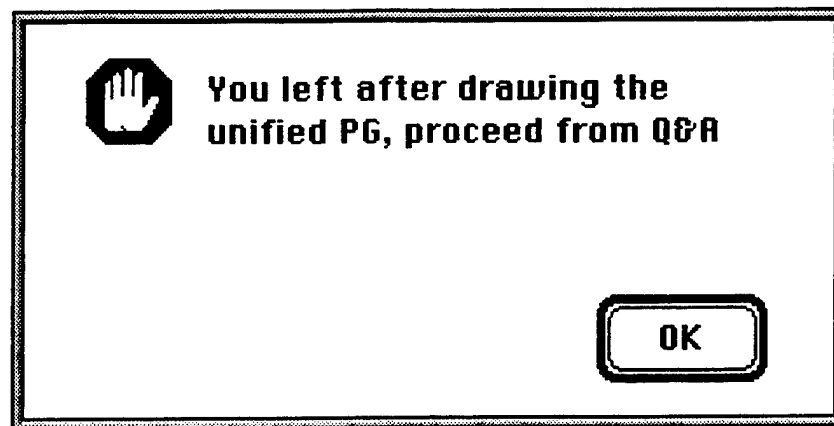


Figure 29. TEMPER1 Window When Re-Starting A Problem

5. TEMPORAL LOGIC / PETRI NET METHODOLOGY: TEMPER2

5.1 Introduction

This section presents an extension the point-interval approach of Section 4 by adding provisions for dates/clock times and time distances for points and intervals. Therefore, the approach sort of combines Dechter's (Dechter et. al. 1991), Kahn and Gory's (1977), and Allen's approaches into a single formalism. The major contribution of this section is the formal tool, called TEMPER2, which automates the inference mechanism of PITL. The tool is based on a graphical representation of the temporal inputs, which not only implements the axiomatic system of PITL, but also verifies system integrity before inference making. Some of the recent approaches that try to combine the qualitative and quantitative aspects of time are due to Kautz and Ladkin (1991), Meiri (1991), and Yao (1994). A final note on TEMPER2's inference engine is that it infers temporal relationships among system intervals/points, calculates time distance among points, and identifies time stamps associated with points, without being engaged in a combinatorially expensive computation. Unlike the approaches of Allen, Kahn and Gory (Kahn and Gory 1977,

Allen 1981) which make use of reference intervals to avoid memory problems, TEMPER2 manages to organize the temporal information with relatively no burden on available storage.

The point interval formalism presented here considers a system's temporal specification on a single time line with a single future. The future of a system is determined by a) the set of events which culminates into the future, and b) the time sequence associated with the occurrence of these events – the definition is similar to the notion of a 'chronicle' by McDermott (1982). A Single Timeline Single Future (STSF) system, therefore, has only one set of events with only one time sequence associated to it – a single chronicle. The time sequence may not be fully specified due to incompleteness in the system specification; there could be events with unspecified temporal relations among them. An STSF system with a fully specified time sequence is shown in Figure 30a.

A Multiple Timelines Multiple Futures (MTMF) system, on the other hand, is characterized either by a single set of events with associated multiple time lines each yielding a different future – type 1, or by multiple set of events with different single/multiple time sequences each representing a total world-history – type 2. Figure 30b presents the two cases of MTMF systems. The treatment of MTMF systems by PITL is beyond the scope of this paper and is left for a future treatment of the subject. Readers who are interested in a detailed discussion on several other topological issues of time are referred to Newton-Smith (1980).

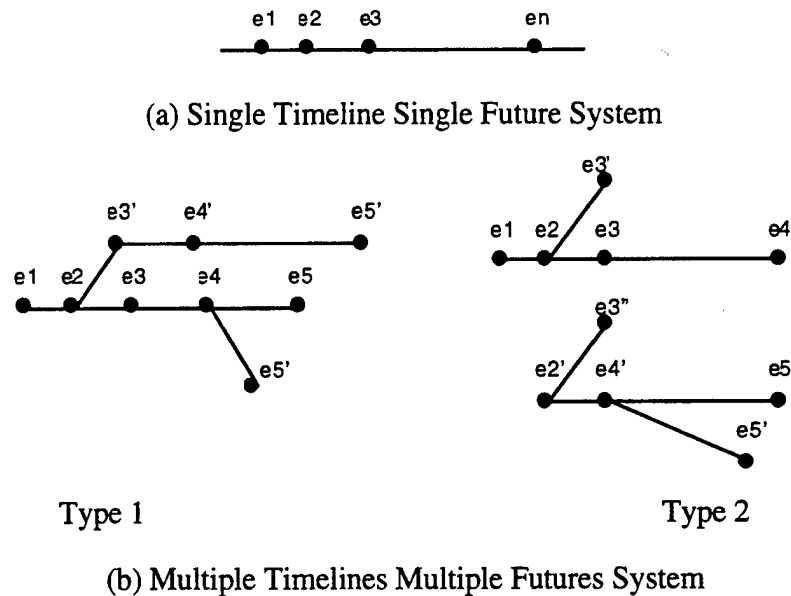


Figure 30 Topology of Temporal Systems

The inference mechanism of PITL, described in Section 4, constructs the analytical representation for the pairs of intervals with unknown temporal relations. The resulting string representation of the relation(s) is pattern matched with the string representations of Figure 14 to infer possible temporal relation(s) between the intervals. An inference engine for PITL, therefore, requires an exhaustive enumeration of the result through all feasible combinations of available statements, provided no knowledge of the system's correctness is available a priori. An inference engine that outputs the result as soon as it finds the first feasible set of inputs can only be applied to a known consistent system of temporal statements. This, in turn, requires a front-end verification mechanism for the PITL statements. This was described in Section 4..

An extension to the PITL formalism allows assignment of actual lengths to intervals, time distances between points, and time stamps to points representing the actual time of occurrences. The approach extends the lexicon of PITL by adding the following two functions to it:

Functions:

length: interval length function that assigns a non-zero positive real number to a system interval, e.g., **length**[X] = d, where $X = [sx, ex]$

time: time stamp function that assigns a real number to a system point, e.g., **time**[p1] = t.

A discussion on the inference mechanism for the extended PITL system is presented in the next section.

5.2 TEMPER2

This section presents the TEMPoral programmER (version 2) (or TEMPER2), which implements the inference mechanism of PITL. The TL/PN approach transforms the system's specifications given by temporal statements into a graph structure. The approach then identifies temporal ambiguities and errors (if present) in the system's specifications by first verifying the system for qualitative temporal relations and then checking the quantitative temporal relations for errors. Once the system is verified for correctness, the temporal inference engine (TIE) infers temporal relations among system's intervals, identifies the *windows* of interest to the user, calculates lengths of intervals and windows, and infers actual time of occurrence of events. The temporal inference engine, TIE, of the TL/PN methodology performs all these tasks by completely

avoiding the combinatorial nature of the inference mechanism. The following sections present a detailed account of all the modules shown in Figure 17.

A. Language

The input to TEMPER2 are statements in PITL. The following Context-free Grammar, CFG(V, T, S, P) represents lexical and syntactical structure of the TEMPER2 inputs.

Set of non-terminals; V = {<temper-input>, <temporal_statement>, <interval>, <temp_relation>, <point>, <temp_relation_1>, <temp_relation_2>, <temp_relation_3>, <letter>, <lower_case_letter>, <upper_case_letter>, <digit>, <sp_ch>}

Set of terminals; T = {Before, Meets, Overlaps, Starts, During, Finishes, Equals, length, time, [,], a, b, ..., z, A, B, ..., Z, 0, 1, 2, ..., 9, (,), _, !, @, \$, %, &, *}

Start Symbol; S = <temper-input>

Set of Productions;

P = {<temper-input> → <temper-input> <temper-input> | <temporal_statement>

<temporal_statement> → <interval> <temp_relation> <interval>

| <interval> <temp_relation_1> <point>

| <point> <temp_relation_2> <interval>

| <point> <temp_relation_3> <point>

| length[<interval>]

| time[<point>],

<interval> → <upper_case_letter> (<letter>|<digit>|<sp_ch>)*

| [<point>, <point>],

<point> → <lower_case_letter> (<letter>|<digit>|<sp_ch>)*

| s<interval> | e<interval>,

<temp_relation> → Before| Meets| Overlaps| Starts| During| Finishes| Equals,

<temp_relation_1> → Before,

<temp_relation_2> → Before| Starts| During| Finishes,

<temp_relation_3> → Before|Equals,

<lower_case_letter> → a|b|...|z,
 <upper_case_letter> → A|B|...|Z,
 <letter> → <lower_case_letter>|<upper_case_letter>,
 <digit> → 0|1|2|...|9,
 <sp_ch> → ()|_||!|@|\$|%|&|*}

In TEMPER2 language, a point is represented by a string of alphanumeric and special characters which starts with a lower-case alphabetic character, e.g., x, y, point, point1, and event are all identifiers representing points. Similarly, an interval is represented by a string of alphanumeric and special characters with its first characters being a upper-case alphabetic character, e.g., X, Y, Interval, Process1, etc. This lexical structure of points and intervals allows the language processor of TEMPER2 to identify semantic errors in the input. A statement 'process1 **Overlaps** Process2' will result in an error, since the identifier 'process1' represents a point and semantically a point (an interval with zero length) can not overlap an interval. Similarly '**length**[event]' will be an erroneous statement because of identifier 'event' being defined as a point. However, 'Process1 **Overlaps** Process2' and '**length**[Event]' will be acceptable temporal statements. The input to TEMPER2 will consist of one or several such acceptable temporal statements connected together with an implicit conjunction. The following is an example of an error-free input to TEMPER2.

Example 15

```

event1 Starts Process1
time[pevent1] = 1000
length[Process1] = 10
event2 Finishes Process1
event2 Starts Process2
length[Process2] = 20

```

B. TL/PN Methodology

The language processor of TEMPER2 performs the lexical and syntactic/semantic analysis on the input statements and reports the errors if found. Once the input is debugged, the TL/PN approach takes each individual statement in the input and transforms it into an equivalent Timed Point Graph (TPG). The TPG representation of a system's temporal aspects organizes the

information contained in temporal statements into a graphical structure. This section presents an analysis based on this graphical representation. The analysis applies certain graph-theoretic concepts on the structure of Timed Point Graphs, identifies their structural properties, and finally interprets the results obtained in terms of the temporal aspects of the systems under consideration.

Definition 28: Timed Point Graph

A Timed Point Graph, $TPG(V, E, D, T)$ is a directed graph with:

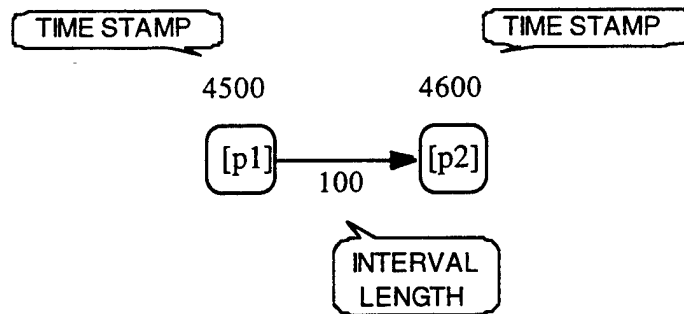
V: Set of vertices with each node or vertex $v \in V$ representing a point on a time line. Two points x and y are represented as a composite point $[x;y]$ if both are mapped to a single point on the time line.

E: Set of edges with each edge $e_{12} \in E$, between two vertices $v1$ and $v2$, also denoted as $(v1, v2)$, representing a temporal relation ' $<$ ' between the two vertices — $(v1 < v2)$.

D: Edge-length function (possibly partial): $V \rightarrow \mathbb{R}$ (set of real numbers)

T: Vertex-time-stamp function (possibly partial): $E \rightarrow \mathbb{R}$

Figure 31 presents a two node Timed Point Graph with time stamps and arc length, and the corresponding temporal situation represented by the TPG. The figure also presents a correspondence between the time stamps and edge lengths: a TPG with only time stamps can be represented by an equivalent PG with edge length expressions and vice versa by using a reference time stamp for the conversion.



Time stamp (time of occurrence) of $p1$ is 4500 time units.

Time stamp of point $p2$ is 4600 time units.

The time distance between two points $p1$ and $p2$ is 100 time units;
 $\text{time}[p2] - \text{time}[p1] = \text{length}[[p1, p2]]$

Figure 31 Timed Point Graph Representation of a Temporal Situation

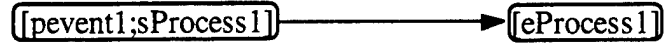
The following examples present the temporal-statement-to-TPG translation process.

Example 16

Consider the TEMPER input presented in Example 15. The TPGs representing each statement are constructed as follows:

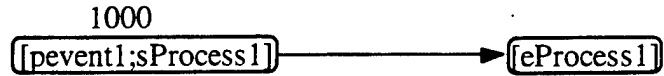
1) event1 **Starts** Process1

Since the identifiers 'event1' and 'Process1' correspond to a point and an interval respectively, the algebraic inequality representing the statement is given as: $pevent1 = sProcess1 < eProcess1$. The PG can now be constructed as:



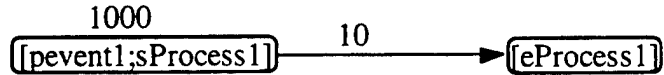
time[pevent1] = 1000

The statement puts a time stamp of '1000' time units on the node with 'pevent1' as one of labels associated with it.



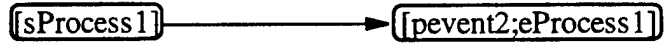
length[Process1] = 10

This statement assigns a length of '10' time units to the edge from 'sProcess1' to 'eProcess1'.

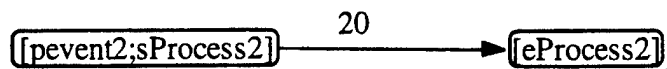


event2 **Finishes** Process1

The inequality representing this temporal statement is: $sProcess1 < eProcess1 = pevent2$. The corresponding TPG will therefore be:



Similarly, Timed Point Graph representing the last two statements, 'event2 **Starts** Process2' and '**length**[Process2] = 20' can be constructed as:



The TPG representing the entire system of temporal statement is then constructed by *unifying* (Definition 28) individual PGs to a (possibly) single connected graph. Figure 32 shows the unified TPG for the temporal statements in Example 17. Note that the unifying process only looks at the labels of the nodes to identify equalities, and does not take into consideration the arc lengths assigned to edges in the TPGs.

Definition 28: Unification

- (a) For all v_i and $v_j \in V$, s.t. $\mathbf{time}[v_i] < \mathbf{time}[v_j]$, construct a directed edge from node v_i to v_j with $\mathbf{length}[[v_i, v_j]] = \mathbf{time}[v_j] - \mathbf{time}[v_i]$;
- (b) Let $v_i = [p_i; \dots; p_n]$ and $v_j = [p_j; \dots; p_m]$ be two nodes in a TPG representation. If there exists a point p_k such that $p_k \in [p_i; \dots; p_n]$ and $p_k \in [p_j; \dots; p_m]$ or $\mathbf{time}[v_i] = \mathbf{time}[v_j]$ then the two nodes are merged into a single composite node ' $v_i; v_j$ ' such that:

$$v_i; v_j = [p_i; \dots; p_n] \cup [p_j; \dots; p_m]$$

$$*v_i; v_j = *v_i \cup *v_j$$

$$v_i; v_j^* = v_i^* \cup v_j^* \quad (\text{where } *v_i \text{ and } v_i^* \text{ represent the pre- and post-set of node } v_i.)$$

$$\mathbf{time}[v_i; v_j] = \mathbf{time}[v_i] = \mathbf{time}[v_j]$$

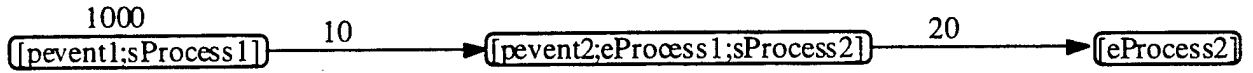


Figure 32 Unified PG Representations for Example 17

System Verification – Phase I

The inference engine of PITL requires a consistent system specification in order to infer temporal relation without enumerating all possible combinations of known temporal relations. Definition 17 has presented a general definition of inconsistency, while Definition 29 characterizes all possible cases of inconsistency in a PITL system.

Definition 29: Inconsistency in PITL

A system's description contains inconsistent information if

- (a) for some intervals X and Y both $X R_i Y$ and $X R_j Y$, $i \neq j$, or $X R_i Y$ and $Y R_j X$ (with the exception of $R_i = R_j = \mathbf{Equal}$) hold true.

or

(b) for a point p_1 , the system calculates two different time stamps.

or

(c) for some points p_1 and p_2 , $p_1 < p_2$, the system can determine two different lengths for the interval $[p_1, p_2]$.

Some of the inconsistent cases, of the type defined in Definition 29b, are trivially detected during the unification process: whenever the system tries to merge two nodes with different time stamps into a single node, it signals an error.

Once a unified Timed Point Graph representation is achieved, the graph is checked for other inconsistent cases defined by Definition 29a. Such inconsistent cases are characterized by Proposition 3, namely, that a set of temporal statements is inconsistent if the PG representation of the set contains self-loops and/or cycles.

Definition 5: *TPG Self-loop*

In a TPG, an arc forms a self-loop if it originates and ends in the same vertex.

A necessary condition for a consistent set of temporal statements has been given by Proposition 4; it also applies in the case of TPGs. Once cycles are detected in a TPG by calculating non-zero S-invariants, the nodes responsible for these cycles can be easily identified. This will, in turn, identify intervals involved in these cycles. This information can be used to correct the system of temporal statements.

C. *Folded Point Graph Representation*

Phase I of the verification process identifies temporal errors present in the system caused by the qualitative input to TEMPER (as described in Section 4 for TEMPER1). The unification process itself identifies some of the erroneous time stamps during the course of unifying individual TPGs. This section presents an analysis, done on the unified TPG, that looks at the edge (arc) length expressions and *folds* the unified TPG into a folded Timed Point Graph. The folding process establishes new temporal relations among system intervals, inferred through the quantitative analysis of the known temporal relations specified by interval lengths and time stamps. The quantitative input provided to TEMPER may in turn have inconsistencies in it. Therefore, a

second phase of system verification is carried out to ensure the correctness of the temporal system before the inference engine is invoked to process queries. A detailed account of the folding process is provided as follows. A correspondence between time stamps and edge length expressions has been presented earlier (Figure 31), therefore, the approach and results are illustrated with TPGs with edge length expression only. The results can be easily applied to graphs with time stamps using the equivalence.

Definition 31: Branch (Join) Node

A vertex $v \in V$ in a Point Graph is termed a Branch (Join) node if it has multiple outgoing (incoming) edges connected to it.

Figure 33 shows a pictorial representation of a branch and a join node in Timed Point Graphs.



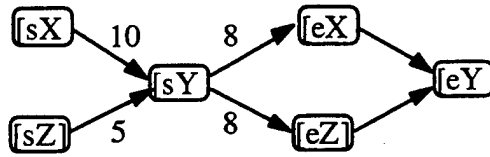
Figure 33 Branch and Join Nodes in Timed Point Graphs

Definition 32: Branch Folding

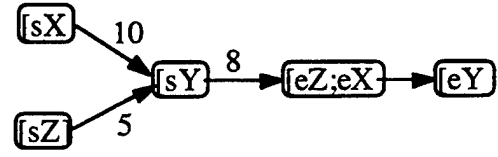
A branch node $v_i \in V$ is said to be folded if, for all v_j and v_k in the post-set of v_i , with:

- **$\text{length}[[v_i, v_j]] < \text{length}[[v_i, v_k]]$** , the edge from v_i to v_k , denoted as (v_i, v_k) , is replaced by an edge (v_j, v_k) with **$\text{length}[[v_j, v_k]] = \text{length}[[v_i, v_k]] - \text{length}[[v_i, v_j]]$** , and the vertex v_k removed from the post-set, or
- **$\text{length}[[v_i, v_j]] = \text{length}[[v_i, v_k]]$** , the two vertices v_j and v_k are merged into a single vertex with composite label ' $v_j;v_k$ ', and **$\text{length}[[v_i, v_j;v_k]] = \text{length}[[v_i, v_k]] (= \text{length}[[v_i, v_j]])$** .

The methodology applies the branch-folding process to all the original and newly created (formed during the folding process) branch nodes in the unified net. Figure 34 illustrates the process by folding a unified TPG.



(a) Unified PG



(b) PG After Branch Folding

Figure 34 Branch Folding

The branch folding process, when applied to all the branch nodes of a graph, yields a partially folded TPG having nodes with at most one outgoing edge with edge length expression. Since all the edges in the TPG may not have edge lengths associated with them, the branch folding may not result in a branch-node-free TPG. Readers may have noted the fact that some of the inconsistent cases, identified in Definition 29, can be detected during the branch folding process; however, a discussion on this issue is delayed for detailed treatment in the next section.

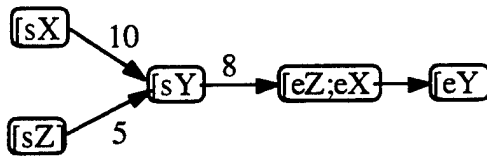
The TPG so obtained is further treated by a join folding process which applies a similar process to all the joins in the graph.

Definition 33: Join Folding

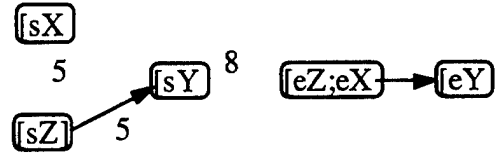
A join node $v_i \in V$ is said to be folded if, for all v_j and v_k in the pre-set of v_i , with:

- $\text{length}[[v_j, v_i]] < \text{length}[[v_k, v_i]]$, the edge (v_k, v_i) , is replaced by an edge (v_k, v_j) with $\text{length}[[v_k, v_j]] = \text{length}[[v_k, v_i]] - \text{length}[[v_j, v_i]]$, and the vertex v_k removed from the pre-set, or
- $\text{length}[[v_j, v_i]] = \text{length}[[v_k, v_i]]$, the two vertices v_j and v_k are merged into a single vertex with composite label ' $v_j;v_k$ ', and $\text{length}[[v_j;v_k, v_i]] = \text{length}[[v_k, v_i]]$ ($= \text{length}[[v_j, v_i]]$).

Figure 35 illustrates the process on the TPG of Figure 34.



(a) Branch-Folded PG



(b) PG After Join Folding

Figure 35 Join Folding

A single application of join folding after a single application of branch folding is all that is required to fully fold the graph. Proposition 9 ensures the fact that single applications of branch folding followed by join folding are enough to fold the graph completely (the term ‘completely’ is used relative to the quantitative information available in the TPG.)

Proposition 9

Let a TPG be folded by branch folding, then a subsequent application of join folding does not create any new branch nodes that can be folded by the branch folding process.

Proof

The branch folding process results in a TPG having nodes with at most one outgoing edge of specified length. The join folding process does not add any new edges to the nodes with lengths expressions. The join folding might result in a new branch node only when it merges the two vertices, v_i and v_j , into a single composite vertex ‘ $v_i;v_j$ ’. This merging, in turn, takes place along the outgoing edges of the original vertices v_i and v_j with edge lengths. The process (Definition 33) replaces the two edges ($v_i, .$) and ($v_j, .$) by a single edge ($v_i;v_j, .$) having the same edge length as that of the replaced ones, leaving the newly created vertex with only one outgoing edge of specified length.

A similar result can be obtained for the branch folding process in terms of join nodes if the TL/PN methodology applies the join folding before branch folding.

System Verification – Phase II

As mentioned earlier, the folding process establishes new temporal relations, among system intervals, inferred through the quantitative analysis of the known temporal relations specified by interval lengths and time stamps. The possible inconsistencies present in the quantitative input to TEMPER2 may hinder the folding process or result in *erroneous structures* of folded graph. Definition 29 identifies the set of possible inconsistencies that might find their way into the temporal system modeled by PITL. The type of inconsistency defined by Definition 29b may reveal itself during the folding process: if during folding a TPG the process finds multiple edges between a branch(join) node and a vertex in its post-(pre-)set, where these edges have different lengths associated with them, then the process halts and reports an error. A pictorial representation of such an erroneous case is presented in Figure 36.

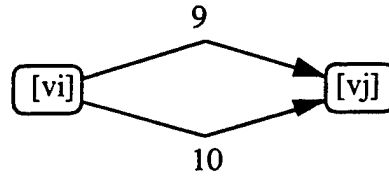


Figure 36 Inconsistent Case Found During Folding Process

During Phase I of the verification process, the unified TPG is checked for acyclicity. The inconsistency, however, can result in creation of new cycles in the graph during the folding process. The following example illustrates the issue.

Example 17

Let a temporal system be described by the following set of TEMPER statements:

event1 Starts Process1	length [Process1] = 3
Process1 Meets Process2	length [Process2] = 3
event1 Before event2	length [[pevent1, pevent2]] = 1
Process2 Before event2	length [eProcess2, pevent3]] = 3
Process2 Before event3	

The corresponding unified TPG is shown in Figure 37.

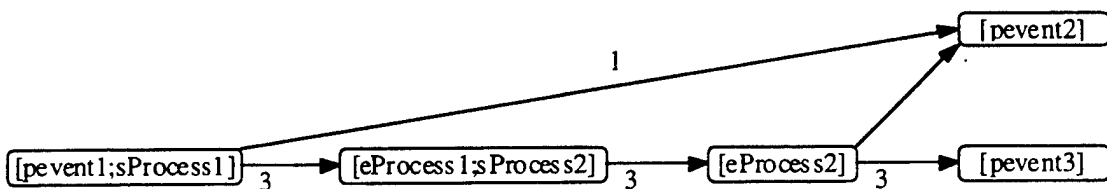


Figure 37 Unified PG for Example 17

The graph has an acyclical structure and, therefore, contains no inconsistent cases that can be identified at the first phase of the verification process.

The branch node labeled 'pevent1;sProcess1' has two outgoing edges with edge lengths and it can be folded by the branch folding procedure. The resulting folded graph, with a cycle, is given in Figure 38.

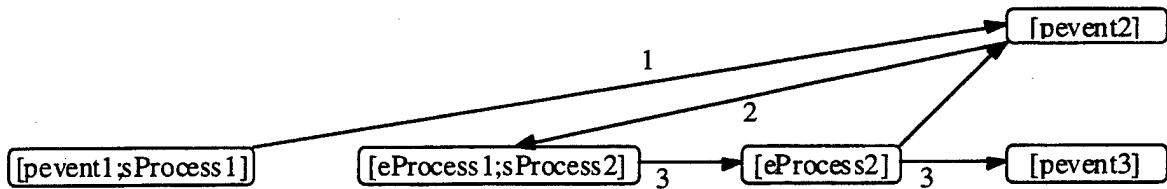


Figure 38 Partially Folded TPG for Example 17

The folded graph in the previous example can not be further folded and an approach similar to the one in Proposition 3 can be applied to identify the cycles present in the folded TPG. The TL/PN methodology, therefore, constructs a Connectivity Matrix of the folded TPG and calculates the S-invariants of the graph. The resulting non-zero S-invariants identify the cycles (inconsistencies) in the temporal system. (Proposition 7) Unlike the case in Example 17, the creation of cycles during the folding process can have serious effects on the graph. Once a cycle is created during the folding process, it tends to attract the remaining vertices in the TPG towards itself. And if the TPG has edge lengths on all its edges the folding process ends up with a folded TPG which has a single cycle with all its vertices *collapsed* into it. The phenomenon is termed 'Black Hole Effect' for the obvious reason. Example 18 illustrates the effect.

Example 18 Black Hole Effect

Let the temporal system presented in Example 17 be augmented with an additional statement:

$$\text{length}[\text{eProcess2}, \text{pevent2}] = 2$$

The PG with this added information is shown in Figure 39.

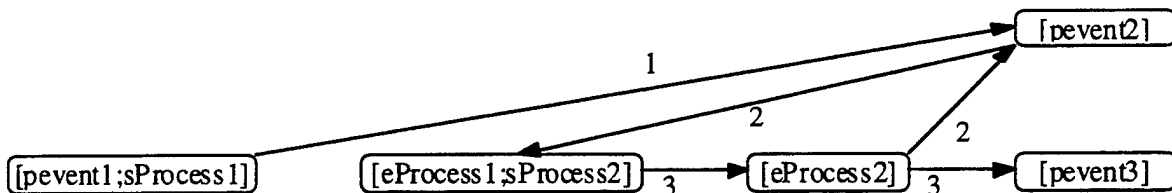


Figure 39 TPG for Example 18

The methodology now finds a branch node (eProcess2) with two outgoing edges that can be folded. The graph after folding the node is shown in Figure 40, part a.

The branch folding proceeds with folding the other branch node (pevent2). The resulting branch-folded TPG is given in part b of Figure 15.

In the previous two iterations of the branch folding procedure, readers must have noticed the fact that the cycle present in the original TPG has caused the vertex labeled as 'pevent3' to wrap around the cycle until it becomes a part of it. A final application of join folding will have a similar effect; the node labeled as 'pevent;sProcess1' will collapse into the black hole and become a part of it, as shown in Figure 40, part c.

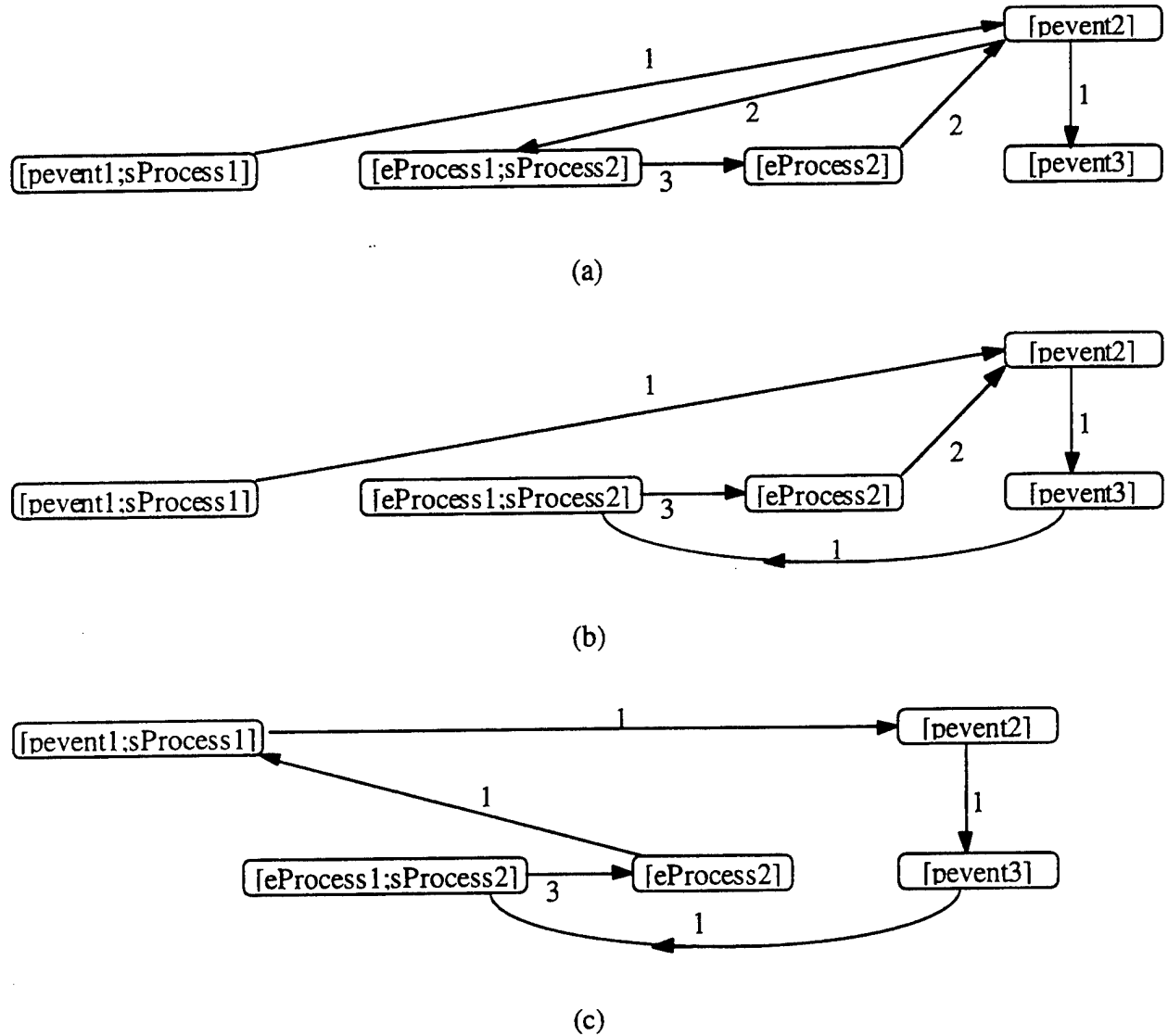


Figure 40 Black Hole Effect

The intensive computational effort required to fold a TPG, and a subsequent loss of it due to the black hole effect demand an earlier detection of cycles during the folding process itself. The folding procedure is, therefore, tailored to identify cycles by assigning dummy time stamps to

vertices being folded: reassignment of a time stamp to an already marked vertex prompts the presence of a cycle.

The folding process folds all the vertices in a PG which are connected together through quantitative temporal relations. A PG with a total Edge-length (D) and/or Vertex-time-stamps function (T) will be folded into a PG having each vertex with at most one incoming and outgoing edge connected to it; the folding process will remove all the branch and join nodes. However, for a PG with partial D and T functions, the folded graph might still have branch and join nodes. The folding process reveals all the quantitative inconsistency in the system; however, some of the inconsistent cases might still be hiding due to the lack of specified edge lengths and/or time stamps on some of the edges and vertices of the graph. A folded TPG with leftover branch and join nodes should be checked for multiple directed paths from any branch node to any other join node. The length expressions corresponding to each such path are equated to each other and the resulting set of equations is checked for feasibility. A set of infeasible equations signals an inconsistent case present in the system. The following example illustrates the issue.

Example 19

Let a temporal system contains the following TEMPER2 statements as part of its system specification:

$$X \text{ During } Y \quad \text{length}[X] = 5 \quad \text{length}[Y] = 4$$

The obvious inconsistency present in the statements neither reveals itself as a cycle nor gets identified during the folding process. However, the error is detected by equating the length expressions corresponding to the two directed paths from vertex 'sY' (branch node) to 'eY' (join node) in Figure 41.

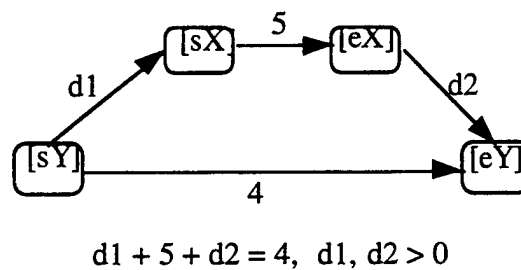


Figure 41 Inconsistent Path Lengths

The TL/PN methodology employs the following steps to carry out this analysis on a folded TPG. The approach makes use of the two search procedures, the *FPSI* and *FPSO* algorithms,

described earlier (Tables 1 and 2). The $FPSO(v)$ algorithm, when applied to an edge v in a TPG, collects all the nodes that have directed paths to v . The $FPSI(v)$ algorithm, on the other hand, collects all the nodes to which v has a directed path. The steps of the methodology are given as follows:

- 1) Construct V_B , the set of all branch nodes in the TPG. Select a node $vi \in V_B$ and calculate PG_i by applying $FPSI(vi)$. Remove from V_B all the branch nodes present in TPG_i .
- 2) Construct V_{ji} , the set of all join nodes in TPG_i . Select a node $vj \in V_{ji}$ and calculate TPG_{ij} by applying $FPSO(vj)$. Remove from V_{ji} all the branch nodes present in TPG_{ij} .
- 3) Merge vi and vj into a single node and calculate S-invariants for the Connectivity Matrix J of the modified TPG_{ij} . The calculated S-invariants correspond to all directed paths from vi to vj . Equate the lengths corresponding to the calculated S-invariants and check the equations for feasibility. If found infeasible, report the error, else iterate through step 2 until there are no elements left in the set V_{ji} .
- 4) Go to step 1 until there are no elements left in the set V_B .

The analysis of the equations constructed as a result of the approach helps bound some of the unknown edge lengths in terms of lower and/or upper limits to their values. The calculated S-invariants can also be used to remove *non-connected chains* in the folded TPG.

D. Temporal Inference Engine (TIE)

The output of TL/PN methodology is a consistent (error free) description of the temporal system represented in terms of a TPG structure. The Temporal Inference Engine (TIE) implements the inference mechanism of PITL to infer the temporal relations among system intervals, time distances among points, length of user-specified intervals, and time stamps associated with points, through a simple search in the TPG.

TIE infers temporal relations between two intervals $X (= [sX, eX])$ and $Y (= [sY, eY])$ by constructing the string representation of the temporal relation (Figure 14) between the two intervals by searching for the directed paths between the vertices representing the intervals in the PG representation. The search for the directed path between two vertices in a TPG uses a *depth-first search* with arc lengths as the heuristic measure; the depth-first search engine first explores the outgoing edge of the current vertex with a length expression. The search, therefore, finds the path

between two vertices which has (possibly) all its constituent edges with length expressions. The sum of all these lengths gives the total distance between the two vertices (points). Similarly, if the time stamp of one of these points is known, the time stamp of the other can be calculated by adding or subtracting the distance (path length) between the two.

The advantage of the TL/PN formalism is that it not only verifies system correctness prior to any inference making, but also overcomes the combinatorial problem associated with inferring new temporal relations; TIE's search engine stops as soon as it finds the first directed path between two vertices under consideration and does not explore all paths between the two vertices to ensure consistency.

TEMPER2's inference engine takes user queries, interprets them, and by invoking the search engine calculates the answers to the input queries. A list of different types of queries that can be processed by TIE is given in Table 5.

Table 5 Queries for Calculating Temporal Relations

Query Structure	Comments
?-R(<interval> <point>, <interval> <point>)	The inference engine returns a specific <i>R_i</i> if it finds one, otherwise it outputs all possible relations that might exist without creating an inconsistency, given incomplete information available.
?-<interval> <temp_relation> <interval> ?-<interval> <temp_relation_1> <point> ?-<point> <temp_relation_2> <interval> ?-<point> <temp_relation_3> <point>	The inference engine returns 'yes' if the temporal relation can be inferred by TIE, 'no' if the inferred relation is in contradiction to the actual relation, and 'plausible' if the input statement is one of the possibilities that can exist in the set of temporal statements without introducing inconsistencies.
?-length[<interval>]	The inference engine returns a real value if the length can be computed, returns upper/lower limit(s) to the value for the length, and 'unknown' if none can be computed.
?-time[<point>]	The inference engine returns a real value if the time stamp for the point can be computed, returns upper/lower limit(s) to the value for the time stamp, and 'unknown' if none can be computed.

6. APPLICATION: Collaborative Air Task Planning

This section illustrates how the graph-based temporal inference engine (TIE) that is used in TEMPER 1 and 2 (Sections 4 and 5) could be applied to Air Task Planning. The section first describes the situation and planning problem and then illustrates how the application of TEMPER can be used to solve that problem.

6.1 Air Task Planning

We assume a military planning situation where there are several planning agents, each with specialized knowledge about a class of air assets for which the agent develops plans. Each agent has responsibility to manage the plans for his assets to ensure that all plan adhere to physical and temporal guidelines. In addition to their individual responsibilities, the agents collectively have the responsibility to create coordinated plans using combinations of the assets to accomplish missions that are assigned to the team of planners. To accomplish these responsibilities, the agents communicate with each other, sharing information needed to build plans that will both satisfy the mission requirements and the individual responsibilities of each agent.

Each agent has local information about the schedule and capabilities of the resources for which he is responsible. The schedule indicates time intervals when each asset is and is not committed to a mission.

The team of planners receives a set of missions to be accomplished. Each mission has requirements that specify the type of action, the time, and the location where the mission is to be performed. The action can be a simple, single action, single asset mission, or an action that will involve several tasks that can be accomplished by several assets of different types. In the latter case, the mission may require that specific temporal relationships be maintained between the actions of different assets for the overall missions is to be successful.

Whenever the team of agents receives a new mission that requires multiple assets for which to develop a plan, the team must determine if a combination of uncommitted assets can be found that will satisfy the mission requirements. To do this, the agents exchange requests and information. Each agent must use a temporal reasoning process to determine if the assets it is responsible for can support the mission requirements. The capabilities of TEMPER can provide

this temporal reasoning process. This dialog between agents and the use of the temporal reasoning via TEMPER can be illustrated with a simple scenario.

6.2 Air Task Planning Scenario

Assume that a team of two agents is required to develop an air tasking plan for a Search and Rescue (SAR) mission. One agent develops plans for Special Operations Forces (SOF) assets and the other agent develops plans for Electronic Surveillance assets, denoted EC assets. The SAR mission requirements are that a SOF asset must ingress to a target area, service that target by picking up a stranded airman, and egress the area, and return to its home base. The mission must be accomplished no later than a specified R hour. In this example R equals 1700 hours on day 23. Because of certain international considerations, the SOF asset must accomplish its mission covertly. This means that the SOF mission must occur concurrently with a mission by an electronic surveillance asset that has the ability to determine if the SOF mission has been detected, and if so, issue a warning to the SOF asset.

Each agent has data on the availability of its assets to perform the mission. The SOF agent has computed the time required for the SOF assets to complete the mission, and has three assets that are capable of performing the mission because they have uncommitted time windows (windows-of-capability) that are large enough to accomplish the task. SOF1 is available from 1200 to 1500, SOF2 is available from 1445 to 1800, and SOF3 is available from 1000 to 1330.

The SOF agent needs to determine which of these assets can meet the temporal mission requirement that the mission be completed by 1700 hours. To do this, the asset availability information of the three SOF assets must be encoded for use in TEMPER. This is accomplished in a two step interval linking procedure. In the first step, the starting and ending time points that define each window-of-capability interval are used to create a pair of temporal relationship statements. For example, the SOF1 interval is encoded with the two statements "sof1s Starts SOF1" and "sof1e Finishes SOF1". For all inputs, the name of an interval is capitalized and the corresponding time points are in lower case. By convention, for time points associated with intervals, the suffix "s" denotes start of an interval and "e" to denote the end of an interval. In step two, additional statements are created that define the temporal relationships between these intervals using their starting and ending points. While there are several ways to do this, the following technique insures that TEMPER will always be able to unambiguously answer queries about the relationship between the intervals. The procedure is to map all of the time points to the real line to

create a total ordering of the time points. Then a set of statements of the form (p_i Before p_{i+1}) are made where i is the index of the total ordering. In the case where $p_i = p_{i+1}$ then the statement (p_i Equals p_{i+1}) is used. Thus, if n is the total number of time points, there will be $n-1$ such statements. In the SOF example there are six time points that define the three intervals. By sorting on the time of the time points, the simple path connecting adjacent points on the real line can be denoted as “sof3s Before sof1s Before sofs3e Before sof2s Before sof1e Before sof2e”. This simple path can be divided into five pair-wise temporal statements. The input to TEMPER is created by combining the statements created in step one and step two as shown in Table 6. The first six entries are created in step one and the remaining five entries are created in step two. TEMPER uses this input to produce the Point Graph shown in Figure 42.

Table 6. Input representing the availability of SOF Assets and Mission Requirements

sof1s Starts SOF1
sof1e Finishes SOF1
sof2s Starts SOF2
sof2e Finishes SOF2
sof3s Starts SOF3
sof3e Finishes SOF3
sof3s Before sof1s
sof1s Before sof3e
sof3e Before sof2s
sof2s Before sof1e
sof1e Before sof2e

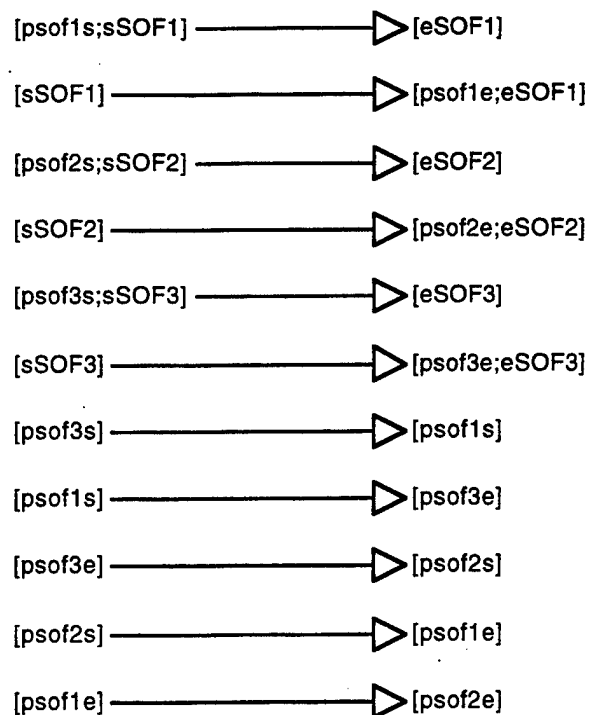


Figure 42 Point Graph Generated by TEMPER

TEMPER then creates the unified point graph which, after some manual rearranging of the vertices, appears as shown in Figure 43. A few enhancements have been added manually to the graph for clarity. First, the three arcs that represent the time each asset is available have been bolded and labeled. A time scale has been added with the time points corresponding to the starting and ending times for the intervals during which each asset is available to perform the mission. Notice that the lighter arcs represent the simple path created in step two of the procedure for interconnecting the intervals.

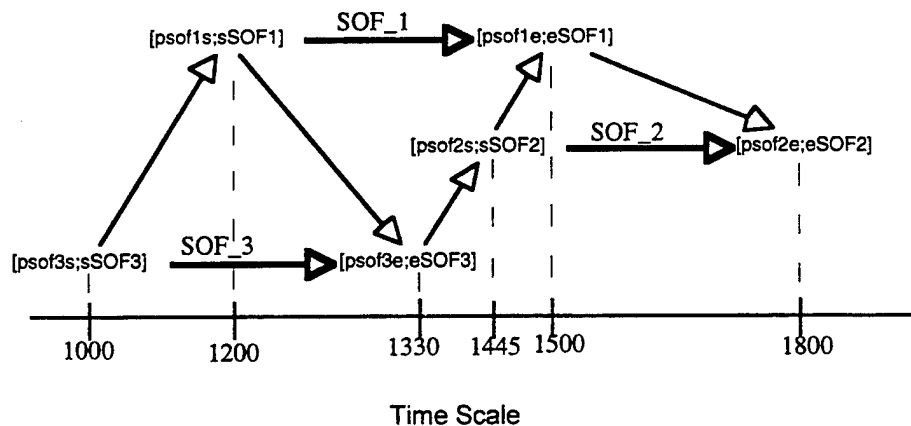


Figure 43. Unified Point Graph of SOF Capability

TEMPER also produces the Incidence Matrix shown in Figure 44. This matrix specifies the ordinary Petri Net shown in Figure 45. Note that the structure of the net is similar to the structure of the unified point graph of Figure 44.

number of places: 8 number of transitions: 6

Trans= [psof1s;sSOF1] [psof1e;eSOF1] [psof2s;sSOF2] [psof2e;eSOF2] [psof3s;sSOF3] [psof3e;eSOF3]

[psof1s;sSOF1]<[psof1e;eSOF1]	1	-1	0	0	0	0
[psof2s;sSOF2]<[psof2e;eSOF2]	0	0	1	-1	0	0
[psof3s;sSOF3]<[psof3e;eSOF3]	0	0	0	0	1	-1
[psof3s;sSOF3]<[psof1s;sSOF1]	-1	0	0	0	1	0
[psof1s;sSOF1]<[psof3e;eSOF3]	1	0	0	0	0	-1
[psof3e;eSOF3]<[psof2s;sSOF2]	0	0	-1	0	0	1
[psof2s;sSOF2]<[psof1e;eSOF1]	0	-1	1	0	0	0
[psof1e;eSOF1]<[psof2e;eSOF2]	0	1	0	-1	0	0

Figure 44 Incidence Matrix Created by TEMPER

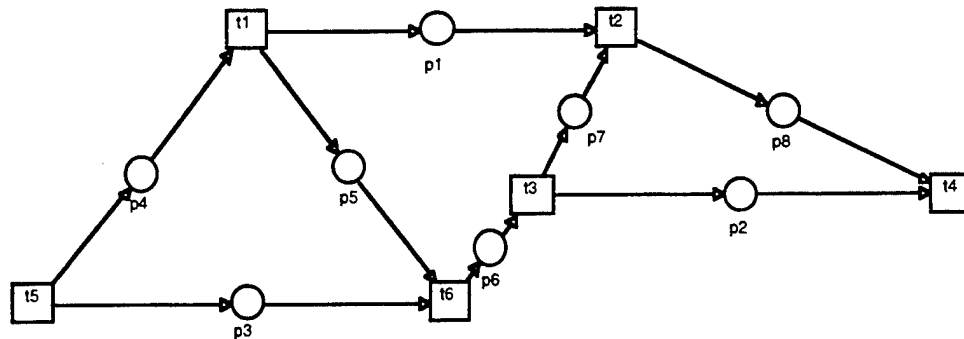


Figure 45. Ordinary Petri Net Representation of Incidence Matrix

The SOF agent now adds the temporal relationships between the mission requirements and the SOF asset availability intervals. Since the mission requirement is that the mission be completed by 1700, he adds the relationship between the mission ending time, “me”, and the nearest SOF asset availability points, the end of the first asset’s availability, SOF1e, and the end of the second asset’s availability, SOF2e. The new input to TEMPER is shown in Table 7.

Table 7. Integrating SOF Asset Availability and Mission Requirements

sof1s Starts SOF1
sof1e Finishes SOF1
sof2s Starts SOF2
sof2e Finishes SOF2
sof3s Starts SOF3
sof3e Finishes SOF3
sof3s Before sof1s
sof1s Before sof3e
sof3e Before sof2s
sof2s Before sof1e
sof1e Before sof2e
sof1e Before me
ms Starts M
me Finishes M
me Before sof2e

Again executing the TEMPER Algorithm results in the Point Graph of Figure 46, the Unified Point Graph of Figure 47, the Incidence Matrix of Figure 48. The Petri Net of Figure 49 can be constructed from the Incidence Matrix. Notice that the Unified Point Graph and the Petri Net contain the graphs of Figures 43 and 45 as sub-nets with the new mission requirement connected to the time points given in the input.

The SOF agent queries TEMPER to find the temporal relationship between each SOF asset and the mission. Because the mission requirement is that the mission be completed by R hour (1700 hours), the query is structured to compare the end of the M interval with the end of each SOF interval. These queries and their results (shown in Table 8) lead to the determination that SOF1 and SOF3 are capable of completing the mission prior to the mission completion time, while SOF2 is not.

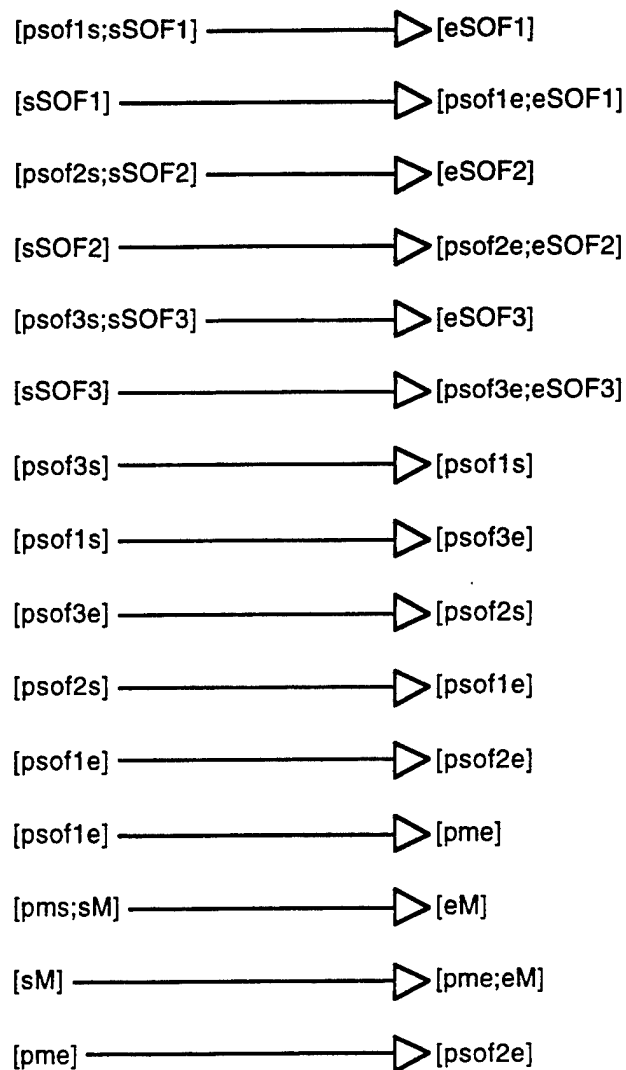


Figure 46 Point Graph of SOF Availability and Mission Requirements

Table 8. Result of Queries to TEMPER by SOF Agent

Query	Response
?- $\Delta([eM, eM], [eSOF1, eSOF1])$	eSOF1 Before eM
?- $\Delta([eM, eM], [eSOF1, eSOF1])$	eM Before eSOF2
?- $\Delta([eM, eM], [eSOF1, eSOF1])$	eSOF3 Before eM

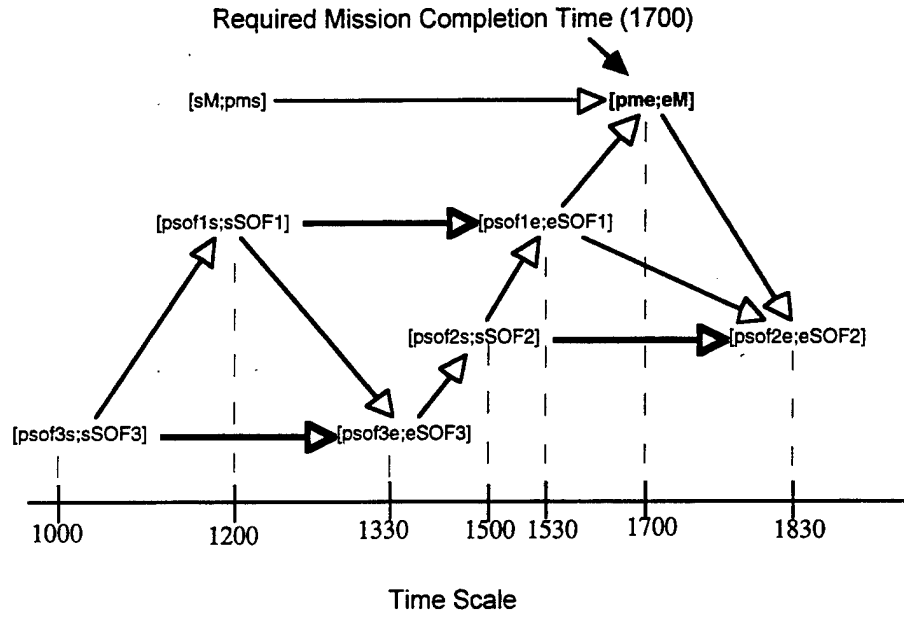


Figure 47 Unified Point Graph of SOF Windows of Capability and the Mission Window of Opportunity

Number of places: 11 number of transitions: 8

Trans= [psf1s;sSOF1] [psf1e;eSOF1] [psf2s;sSOF2] [psf2e;eSOF2] [psf3s;sSOF3] [psf3e;eSOF3] [pme;eM] [sM;pms]

[psf1s;sSOF1]<[psf1e;eSOF1]	1	-1	0	0	0	0	0	0
[psf2s;sSOF2]<[psf2e;eSOF2]	0	0	1	-1	0	0	0	0
[psf3s;sSOF3]<[psf3e;eSOF3]	0	0	0	0	1	-1	0	0
[psf3s;sSOF3]<[psf1s;sSOF1]	-1	0	0	0	1	0	0	0
[psf1s;sSOF1]<[psf3e;eSOF3]	1	0	0	0	0	-1	0	0
[psf3e;eSOF3]<[psf2s;sSOF2]	0	0	-1	0	0	1	0	0
[psf2s;sSOF2]<[psf1e;eSOF1]	0	-1	1	0	0	0	0	0
[psf1e;eSOF1]<[psf2e;eSOF2]	0	1	0	-1	0	0	0	0
[psf1e;eSOF1]<[pme;eM]	0	1	0	0	0	0	-1	0
[sM;pms]<[pme;eM]	0	0	0	0	0	0	-1	1
[pme;eM]<[psf2e;eSOF2]	0	0	0	-1	0	0	1	0

Figure 48 Incidence Matrix Output of TEMPER

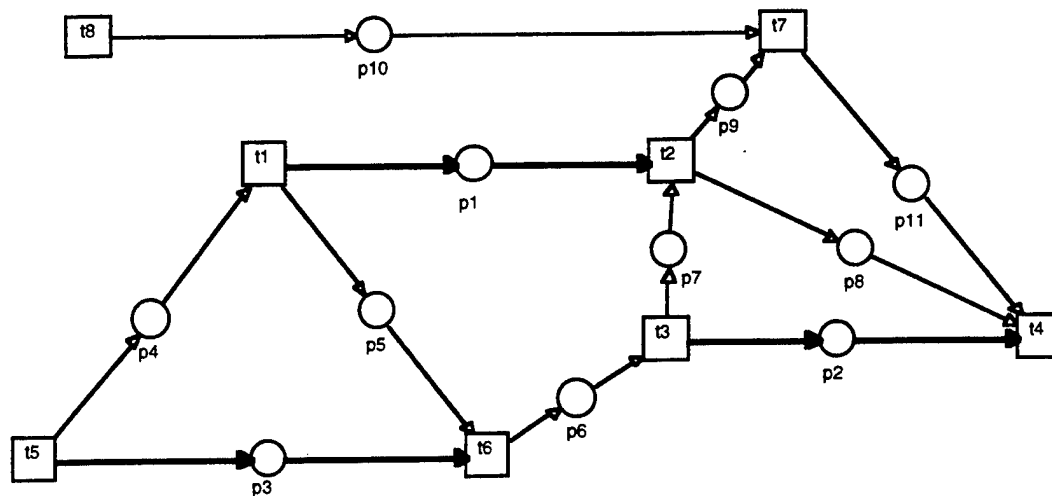


Figure 49. Ordinary Petri Net Representation of Incidence Matrix

As a result of the query of TEMPER, the SOF agent can extract from Table 9, the portion of the of the temporal relationships that meets the mission requirements, that is the statements that define the windows of capability for SOF1 and SOF3.

Table 9. Point/Interval Relationships Defining Windows of Capability for SOF Assets That Can Meet Mission Requirements

sof1s Starts SOF1
sof1e Finishes SOF1
sof3s Starts SOF3
sof3e Finishes SOF3
sof3s Before sof1s
sof1s Before sof3e
sof3e Before sof1e

The SOF agent sends the final set of Point/Interval relationships and the start and end times for each interval that define the SOF asset windows of capability that meet the SAR mission requirements (the first four entries in Table 7) to the electronic surveillance agent as a tasking. This tasking contains the requirement that the EC assets must perform a supporting mission such that

the SOF mission interval will occur *During* the EC Asset mission. Note that the electronic surveillance agent is being asked to support only one of the two potential SOF missions.

In this scenario, assume the Electronic Surveillance agent has three assets available during the following intervals: EC1 0900 to 1300, EC2 1430 to 1800, EC3 1130 to 1530. Using the interval linking procedure described for the SOF agent, this availability is encoded as a TEMPER input as shown in Table 10. The corresponding Unified Point Graph is shown in Figure 50.

Table 10 Point/Interval Relationships Defining Windows of Capability input for EC Assets

ec1s Starts EC1
ec1e Finishes EC1
ec2s Starts EC2
ec2e Finishes EC2
ec3s Starts EC3
ec3e Finishes EC3
ec1s Before ec3s
ec3s Before ec1e
ec1e Before ec2s
ec2s Before ec3e
ec3e Before ec2e

The Electronic Surveillance agent creates a new input to TEMPER by taking the interval specifications for the EC assets (the first six entries of Table 8) and adding to them the list from the SOF agent . This produces the list in Table 11. Using the times for all of the staring and ending points, the agent uses Step Two of the interval linking procedure described for the SOF agent to combine and link all EC and SOF intervals. This results in Table 12.

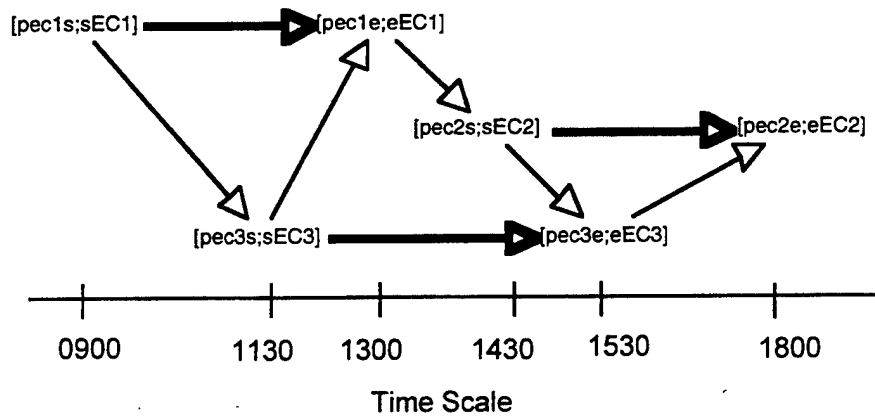


Figure 50. Unified Point Graph EC Assets With individual Time Scales

Table 11. Combine EC and SOF Availability Intervals

ec1s Starts EC1
ec1e Finishes EC1
ec2s Starts EC2
ec2e Finishes EC2
ec3s Starts EC3
ec3e Finishes EC3
sof1s Starts SOF1
sof1e Finishes SOF1
sof3s Starts SOF3
sof3e Finishes SOF3

When given Table 12 as an input, TEMPER will produce the Point Graph of Figure 51.

After unifying the point graph of Figure 51, and with some manual re-arranging of the vertices, the unified point graph of Figure 52 is created from the one created by TEMPER. As before, a time scale has be added for clarity and the arcs have been highlighted to denote the intervals of availability of each of the assets. The effect of step two of the interval linking

procedure shown with the plain arcs. As before, the result is that the graph contains a simple path from the earliest time point vertex to the latest time point vertex that traverses all vertices. This ensures that TEMPER can answer unambiguously any query.

Table 12 Combined SOF and EC Temporal Relationship Input

ec1s Starts EC1
ec1e Finishes EC1
ec2s Starts EC2
ec2e Finishes EC2
ec3s Starts EC3
ec3e Finishes EC3
sof1s Starts SOF1
sof1e Finishes SOF1
sof3s Starts SOF3
sof3e Finishes SOF3
ec1s Before sof3s
sof3s Before ec3s
ec3s Before sof1s
sof1e Before ec1e
ec1e Before sof3e
sof3e Before ec2s
ec2s Before sof1e
sof1e Before ec3e

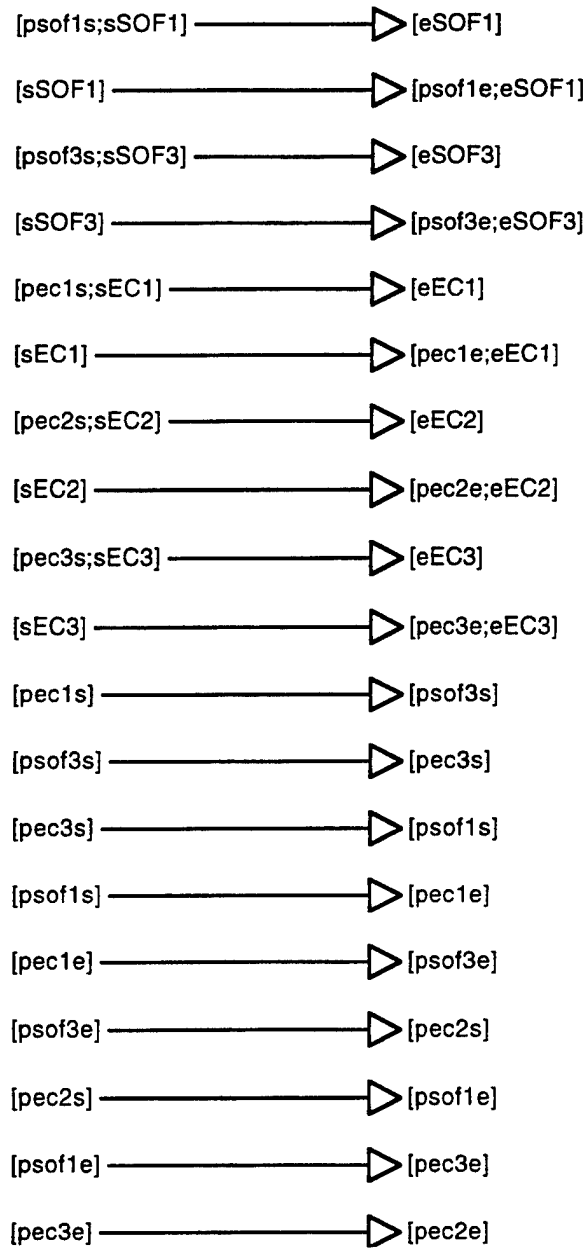


Figure 51 Point Graph of Combined SOF and EC Windows of Capability

The Electronic Surveillance agent is now ready to determine which assets can satisfy the task requirement that the SOF mission occur *During* the EC mission. Six queries of the form $\Delta([sEC1, eEC1], [sSOF1, eSOF1])$ to test the relationship between the two SOF assets and the three EC assets intervals (windows of capability) yields the following results (Table 13):

Table 13 Result of EC Agent Query To TEMPER

Query	Response
?- $\Delta([sEC1, eEC1], [sSOF1, eSOF1])$	EC1 Overlaps SOF1
?- $\Delta([sEC2, eEC2], [sSOF1, eSOF1])$	SOF1 Overlaps EC2
?- $\Delta([sEC3, eEC3], [sSOF1, eSOF1])$	SOF1 During EC3
?- $\Delta([sEC1, eEC1], [sSOF3, eSOF3])$	EC1 Overlaps SOF3
?- $\Delta([sEC2, eEC2], [sSOF3, eSOF3])$	SOF3 Before EC2
?- $\Delta([sEC3, eEC3], [sSOF3, eSOF3])$	SOF3 Overlaps EC3

Only the third query yields the temporal relationship *During* as required by the mission, thus the solution to the planning problem is to use SOF1 from 1200 to 1500 hours and EC3 from 1130 to 1530 hours. Based on this result, the Electronic Surveillance agent can finalize the plan selecting the set of temporal statements that pertain only to the availability intervals for SOF1 and EC3. The result of the query, SOF1 During EC3, is added to these statements to create the input in Table 14.

Table 14 Final Input to TEMPER

sof1s Starts SOF1
sof1e Finishes SOF1
ec3s Starts EC3
ec3e Finishes EC3
SOF1 During EC3

TEMPER creates the Point Graph , the Unified Point Graph , and the Incidence Matrix of Figures 54, 55, and 56, respectively.

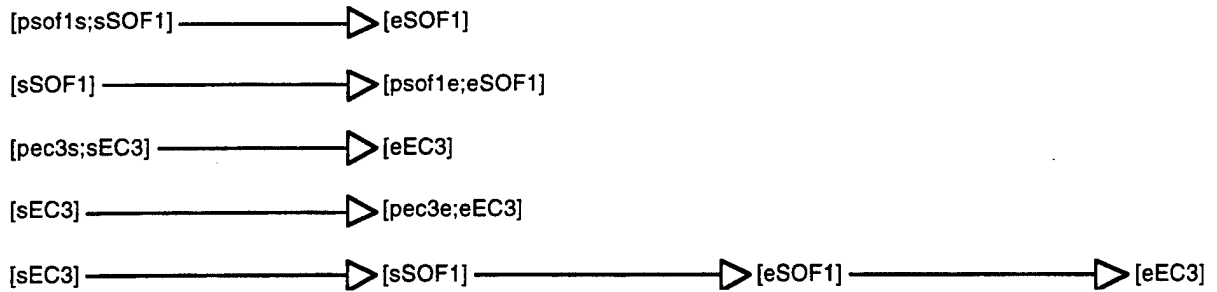


Figure 54 Point Graph of the Final Solution to the Planning Problem

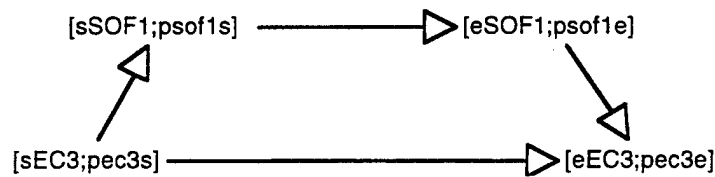


Figure 55. Unified Point Graph of the Final Solution

number of places: 4 number of transitions: 4

Trans= [sSOF1;psof1s] [eSOF1;psof1e] [sEC3;pec3s] [eEC3;pec3e]

[sSOF1;psof1s]<[eSOF1;psof1e]	1	-1	0	0
[sEC3;pec3s]<[eEC3;pec3e]	0	0	1	-1
[sEC3;pec3s]<[sSOF1;psof1s]	-1	0	1	0
[eSOF1;psof1e]<[eEC3;pec3e]	0	1	0	-1

Figure 56. Incidence Matrix of the Final Solution

Alternatively, the agent could have used the interval linking procedure to produce the following input to TEMPER (Table 15).

Table 15. Final Query to TEMPER Using Interval Linking Procedure

sof1s Starts SOF1
sof1e Finishes SOF1
ec3s Starts EC3
ec3e Finishes EC3
ec3s Before sof1s
sof1e Before ec3e

Given this input, TEMPER would create the Point Graph of Figure 57. This will result in the same Unified Point Graph and Incidence Matrix shown in Figures 55, and 56, respectively.

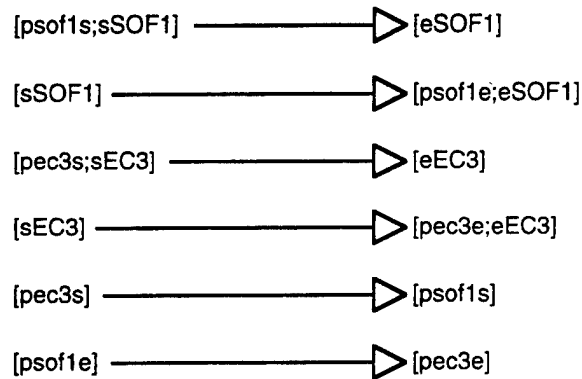


Figure 57. Alternative Point Graph of Final Planning Solution

The Petri Net specified by the final planning solution Incidence Matrix is shown in Figure 58. The places and the transitions are labeled with the names given in the Incidence Matrix. By changing the labels, this Petri Net can be transformed into a more understandable executable model of the final plan and shown in Figure 59. Note that the arcs connecting the two asset time-lines now represent synchronizing coordination between the assets. This synchronization has been

made explicit by indicating that EC3 notifies SOF1 when it has started its mission. SOF1 should not start its mission until this message is received. SOF1 sends a message to EC3 when it is finished its mission, telling EC3 it is okay to end the electronic surveillance mission.

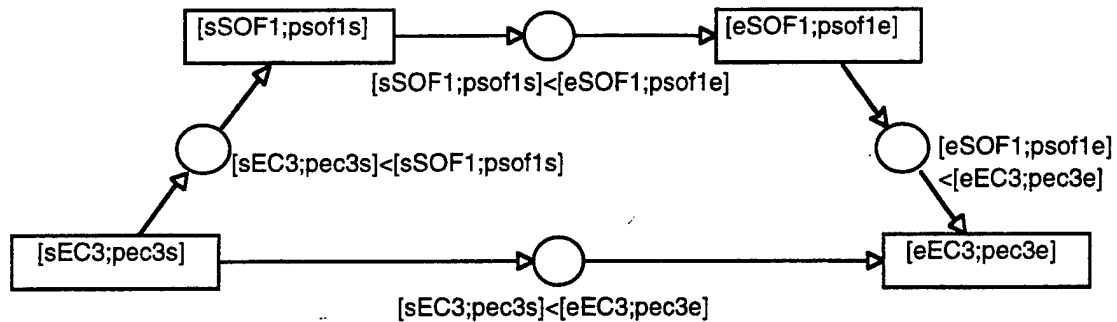


Figure 58. Petri Net of the Incidence Matrix of the Final Planning Solution

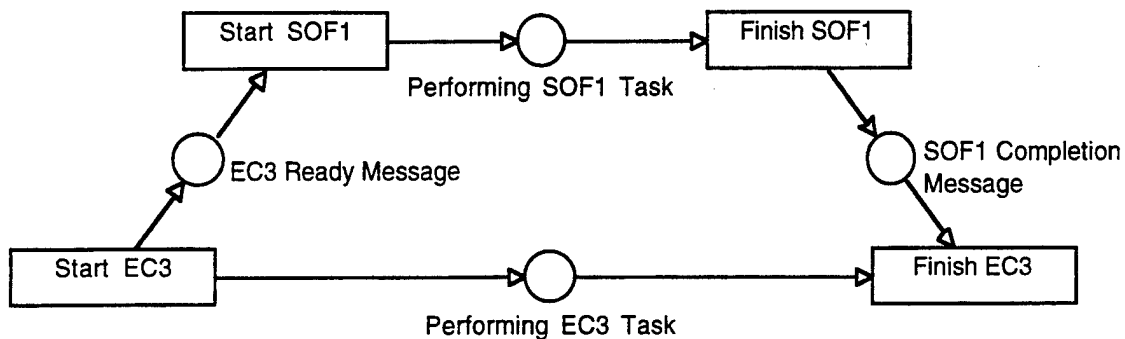


Figure 59. Petri Net Converted to Final Plan

6.3 Summary

We have shown how a temporal reasoning system such as TEMPER can be used to support the development of military plans by a group of planning agents. We presented a simple algorithm that will generate the input to the TEMPER program given a set of time intervals that specify the windows of capability of assets needed for a mission and the interval during which the mission is to be performed (the mission window of opportunity). Each interval is defined by a starting point and an end point and their associated times. This procedure produces an input to TEMPER which in turn produces a unified point graph that will yield unambiguous answers to

queries about the temporal relationship between intervals, points and intervals, and points. These queries can be used to test and find solutions to mission temporal requirements. We illustrated how a team of planners can collaborate using this capability to generate plans given a mission requirement. The illustration was elementary. Clearly real world planning situations will require much more interaction between planning agents. Furthermore the example did not address the issues of selecting between competing alternatives at either the agent level or at the team level. However, one of the main attributes of any planning alternative will be the temporal relationship requirements that are maintained between assets as they perform their tasks.

7. CONCLUSION

The TL/PN formalism provides a basic engine for system description and modeling. The Petri net model of the system is not merely a different representation of the same information present in the statements of point-interval logic, but it provides an analytical tool for temporal reasoning, validation, verification, and calculation of windows of capabilities.

The current approach offers:

- (1) A general, complete and sound formalism of point-interval logic. This extends the approach of Allen's logic.
- (2) An inference engine (TIE) that infers the temporal relationship among intervals without running into combinatorial problem.
- (3) A verification mechanism for the temporal formalism that identifies inconsistencies and incompleteness in the system of temporal statements.

The input statements to the methodology are all connected together with logical AND. Therefore, they represent a system's description on a single time line. Due to incompleteness we may not map these intervals on a single line, but a complete system will have all its intervals on a single time line (a PG with one connected chain.) For any point on this time line there will exist only one future.

The present approach has been extended to take into consideration the lengths of intervals and the actual time (clock time) of occurrence of events. The inference engine has been extended to answer queries regarding actual time or length of an event.

The major extension to this approach currently under way is to consider system description with multiple time lines and multiple futures (MTMF). The temporal statements in such a system could also be presented with logical OR. The TIE will then be extended to plan events (sequences of events) that lead to a particular future state of the system. Here one possible future state (possibly partially specified) can be reached through several time lines. The constraints (mission requirements) will determine the feasible time line(s) to be selected among these alternatives. The approach can be extended to incorporate statements like "Process P takes at least 10 minutes to complete" not currently handled by the approach. The third version of the algorithm, TEMPER3, is expected to address the MTMF problem.

REFERENCES

- Allen, J. F. (1981a) A general model of action and time. Technical report, University of Rochester, TR97
- Allen, J. F. (1981b) An interval based representation of temporal knowledge. In *Proc. 7th IJCAI*,
- Allen, J. F. (1983), Maintaining knowledge about temporal intervals. *CACM*, 26 (11), 832-843.
- Allen, J. F. (1984), Towards a general theory of action and time. *Artificial Intelligence*, 23, 123-154.
- Allen, J. F. and P. J. Hayes (1985a). A Common-sense Theory of Time. *Proc. 9th Int. Joint Conf. on AI*, pp. 528-531.
- Allen, J. F. and P. J. Hayes. (1985b) A commonsense theory of time: the longer paper. Technical report, University of Rochester.
- Dechter, R., I. Meiri, I., and J. Pearl (1991). Temporal Constraint Network. *Artificial Intelligence*, vol. 49, pp. 61-95.
- Galton, A. (1990) Logic for information technology. John Wiley & Sons, Chichester.
- Hillion, H. P. (1986) Performance evaluation of decisionmaking organizations using timed Petri nets. LIDS-TH-1590, Laboratory of Information and Decision Systems, MIT.

Jin, V. Y. (1986) Delays for distributed decisionmaking organizations. LIDS-TH-1459, Laboratory for Information and Decision Systems, MIT, Cambridge.

Kahn, K., and Gory, G. 1977. Mechanizing Temporal Knowledge. *Artificial Intelligence*, vol. 9, pp. 87-108.

Kautz, H. and Ladkin, P. (1991). Integrating Metric and Qualitative Temporal Reasoning. *Proc. of AAAI-91*, pp. 241-246, Anaheim, CA.

Meiri, I. (1991). Combining Qualitative and Quantitative Constraints in Temporal Reasoning. *Proc. of AAAI-91*, pp. 260-267, Anaheim, CA.

MATRA CAP Systemes (1994). Eagle Vision Data Acquisition Segment Technical Brief. S01/PhR-ef/94/061 CD22. Space Observation Division..

Perdu D. M., S. L. Hearold & A. H. Levis (1994). Effectiveness of Two Modes of Information Pull in the Copernicus Architecture, A. H. Levis & I. S. Levis (Editors), *The Science of Command and Control: Part III, Coping with Change*, pp 57-74. Fairfax, VA: AFCEA International Press.

Yao, Y. (1994) A Petri Net Model for Temporal Knowledge Representation and Reasoning. *IEEE Transactions on SMC*, vol. 24, no. 9, 1374-1382.

Zaidi, A. K. (1992) On the generation of multilevel distributed intelligence systems using Petri nets. Technical report, GMU/C3I-112-TH, Center of Excellence in C3I, George Mason University.

Zaidi, A. K. and A. H. Levis (1995) On Verifying inferences in an Influence diagram. *In Proc. 1995 First International Symposium on Command and Control Research and Technology*, National Defense University, Washington, DC, June 19-23, 1995. pp. 443-451.

Zaidi, A. K. and A. H. Levis (1997) Validation and verification of decision making rules. *Automatica*, Feb, 1997